UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
CAMPUS D'ALCOI

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Alcoy

Solución basada en aprendizaje profundo para el reconocimiento de lenguaje de signos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Tercero Nueda, Rubén

Tutor/a: Silvestre Blanes, Javier Lidiano

Cotutor/a: Pérez Llorens, Rubén

CURSO ACADÉMICO: 2022/2023

Solución basada en aprendizaje profundo para el reconocimiento de lenguaje de signos

En España alrededor del 2,3% de la población total sufre algún tipo de discapacidad auditiva. Además, la OMS señala que los casos de sordera irán en aumento cada año.

Este Trabajo Final de Grado, tiene como objetivo crear un algoritmo capaz de identificar y reconocer un subconjunto del lenguaje de signos español.

A partir de un conjunto de datos desarrollado desde cero, entrenaremos al algoritmo mediante técnicas de aprendizaje profundo. Una vez entrenado, desplegaremos una aplicación donde el algoritmo será capaz de reconocer y traducir en tiempo real el signo dado por el usuario.

Si el algoritmo tiene éxito, ayudará a las personas sordas a comunicarse mejor mediante aplicaciones de visión por computador. Además de un apoyo para las personas que intentan aprender el lenguaje de señas. La introducción de estrategias basadas en reconocimiento visual tiene el potencial de mejorar la comunicación en aquellas personas con discapacidad auditiva, causando impactos beneficiosos en diferentes aspectos de sus vidas.

Palabras clave: Aprendizaje Profundo, Pérdida auditiva, NVIDIA Jetson Nano, OpenCV, Tensorflow

Deep learning-based solution for sign language recognition

In Spain, about 2.3% of the total population suffers from some type of hearing impairment. In addition, the WHO states that the number of deafness cases will increase every year.

This Final Degree Project aims to create an algorithm capable of identifying and recognizing a subset of Spanish sign language.

From a dataset developed from scratch, we will train the algorithm using deep learning techniques. Once trained, we will deploy an application where the algorithm will be able to predict in real time the sign given by the user.

If the algorithm is successful, it will help deaf people to communicate better through computer vision applications. In addition to support for people trying to learn sign language. The introduction of strategies based on visual recognition has the potential to improve communication in those with hearing impairment, causing beneficial impacts in different aspects of their lives.

Key words: Deep Learning, Hearing loss, NVIDIA Jetson Nano, OpenCV, Tensorflow

Solució basada en aprenentatge profund per al reconeixement de llenguatge de signes

A Espanya, al voltant del 2,3% de la població total pateix algun tipus de discapacitat auditiva. A més, l'OMS assenyala que els casos de sordera aniran en augment cada any.

Aquest Treball Final de Grau, té com a objectiu crear un algorisme capaç d'identificar i reconèixer un subconjunt del llenguatge de signes espanyol.

A partir d'un conjunt de dades desenvolupat des de zero, entrenarem a l'algorisme mitjançant tècniques d'aprenentatge profund. Una vegada entrenat, desplegarem una aplicació on l'algorisme serà capaç de predir en temps real el signe donat per l'usuari.

Si l'algorisme té èxit, ajudarà les persones sordes a comunicar-se millor mitjançant aplicacions de visió per computador. A més d'un suport per a les persones que intenten aprendre el llenguatge de senyals. La introducció d'estratègies basades en reconeixement visual té el potencial de millorar la comunicació en aquelles persones amb discapacitat auditiva, causant impactes beneficiosos en diferents aspectes de les seues vides.

Paraules clau: Aprenentatge Profund, Pèrdua auditiva, NVIDIA Jetson Nano, OpenCV, Tensorflow

Agradecimientos

Quiero expresar mi más sincero agradecimiento a todas las personas que me han alentado y ayudado en la redacción de este Trabajo Final de Grado. Quisiera destacar especialmente a aquellos que han dejado una huella significativa en mi camino académico y personal:

En primer lugar, mi tutor Javier Lidiano Silvestre Blanes, mi cotutor Rubén Perez Llorens, quienes me han brindado una invaluable formación y apoyo a lo largo de mis años en la universidad.

También quiero agradecer a mi madre, quien ha fomentado mi curiosidad y pasión por comprender el sentido de las cosas, y a mi padre, cuyo espíritu trabajador me ha inspirado constantemente.

Agradezco profundamente a mis amigos, quienes siempre han estado a mi lado con buenos consejos y con un afán inquebrantable por ayudarme en cada paso del camino.

No puedo olvidar mencionar a todas esas personas que, de manera indirecta, me han brindado enseñanzas y aprendizajes valiosos, y han contribuido a forjar la persona que soy hoy.

Y finalmente, a mi amada pareja y compañera de viaje, Arantxa, quiero expresar mi gratitud eterna por su apoyo incondicional y su infinita inspiración.

A cada uno de ustedes, les agradezco de corazón por su contribución y por ser parte de mi éxito en este importante logro académico. Estoy verdaderamente agradecido por su influencia en mi vida.

Índice

Índice de Tablas

Índice de Fórmulas

Índice de Figuras

1. Introducción

La comunicación es fundamental para las personas con discapacidad auditiva. El **reconocimiento del lenguaje de signos** representa un campo de estudio que aspira a potenciar su interacción con el entorno. El **Deep Learning**, una rama de la inteligencia artificial utiliza redes neuronales artificiales para aprender características complejas de los datos. Esta técnica es efectiva en tareas como el reconocimiento de patrones, visión por computador y procesamiento de lenguaje natural.

En el caso del reconocimiento de signos, el aprendizaje profundo ofrece la capacidad de analizar e interpretar los gestos de las manos.

A lo largo de este trabajo, se explorará cómo el aprendizaje profundo puede contribuir al reconocimiento del lenguaje de signos, abordando desafíos como la variabilidad de los gestos y las diferencias regionales en los sistemas de signos. Además, se construirá un sistema de interpretación en tiempo real.

En el campo de la visión artificial, la detección de poses y manos ha sido objeto de investigación durante varias décadas y ha generado gran interés en la comunidad informática. En este estudio, se han aprovechado las técnicas de **transfer learning** y **data augmentation** para hacer posible el desarrollo de este.

Para el entrenamiento del modelo de detección de signos, se ha utilizado una NVIDIA Jetson Nano de 2 GB. Esta plataforma ha demostrado ser más que suficiente para lograr un entrenamiento preciso de la red neuronal, proporcionando los recursos necesarios para procesar y analizar los datos de entrada.

1.1. Motivación

La decisión de realizar este proyecto surge de múltiples razones, pero, sobre todo, se trata de un reto personal. El **reconocimiento dactilológico del Lenguaje de Signos Español** [1] es una herramienta esencial que **permite a las personas sordas interactuar de manera más eficaz con el mundo que les rodea**.

El reconocimiento de la dactilología del Lenguaje de Signos Español es un desafío que va más allá de lo meramente académico. A pesar de requerir la aplicación de conocimientos adquiridos en asignaturas como Visión por computador, Machine Learning y programación, este proyecto es principalmente un desafío personal. Mi objetivo es poner en práctica los conceptos que he aprendido a lo largo de mi formación en Ingeniería Informática, integrándolos en una solución real.

1.2. Objetivos

El objetivo principal de este Trabajo Final de Grado es desarrollar un algoritmo de reconocimiento en tiempo real de la dactilología utilizada en el Lenguaje de Signos Español con alta precisión y confiabilidad. El propósito fundamental es permitir que el algoritmo sea capaz de **reconocer y traducir de manera efectiva las diferentes letras del alfabeto de signos**. Además, se busca que el algoritmo tenga la capacidad de construir palabras completas a partir de las letras reconocidas y realizar la interpretación correspondiente al lenguaje de signos, ya sea en texto escrito o voz.

Además, se propone desarrollar un traductor, destinado a asistir a aquellas personas interesadas en aprender el lenguaje de signos o que deseen establecer una comunicación efectiva con personas con discapacidad auditiva y de habla.

1.3. Impacto Esperado

En el **mundo empresarial**, se han gestado múltiples soluciones innovadoras para enfrentar los desafíos asociados con el lenguaje de signos. Un ejemplo es **SignAll**, un startup que ha desarrollado un dispositivo singular capaz de facilitar la comunicación entre una persona sorda y una oyente mediante la traducción automática del **ASL (Lenguaje de Signos Americano).**

Paralelamente, en **Europa**, ha habido avances significativos en este ámbito, particularmente con la introducción de una aplicación que reproduce de manera digital las grabaciones de intérpretes reales comunicándose a través del lenguaje de signos.

Estos dos proyectos, que se examinarán con más detalle posteriormente, son solo ejemplos de las múltiples soluciones que existen. Sin embargo, a pesar de la variedad de aplicaciones disponibles, se observa una carencia destacable de recursos específicamente diseñados para el Lenguaje de Signos Español.

Este proyecto es, por tanto, más que un desafío académico: es una **oportunidad personal para aplicar los conceptos y habilidades** que se han adquirido a lo largo de mis estudios en el **grado en Ingeniería Informática**.

1.4. Metodología

La **metodología** propuesta para la creación del algoritmo deberá seguir los siguientes pasos:

1. Recopilación y preparación del conjunto de datos.
2. Anotación y etiquetado de los datos.
3. Preprocesamiento de datos.
4. Selección del modelo neuronal a utilizar.
5. Diseño y entrenamiento del modelo neuronal.
6. Evaluación y ajuste del modelo.
7. Validación y pruebas en tiempo real.
8. Mejora continua y refinamiento.

A través de estos pasos, se buscará garantizar la calidad y representatividad de los datos, seleccionar y diseñar una red neuronal adecuada, entrenarla con el conjunto de datos anotado, evaluar y ajustar el modelo para obtener resultados precisos y robustos. Luego, se realizará pruebas en tiempo real para verificar su funcionamiento en entornos reales.

1.5. Estructura

Este trabajo Final de Grado, se **estructura** en varios apartados destinados a describir una **solución de aprendizaje profundo para el reconocimiento de la dactilología del Lenguaje de Signos Español**:

- **Estado del arte:** Se proporciona una revisión de la literatura y las investigaciones más relevantes en el campo del reconocimiento del Lenguaje de Signos y la aplicación de diferentes técnicas y herramientas para su desarrollo. Además, se exponen los diversos proyectos que se han desarrollado acerca de esta problemática.
- **Análisis del problema:** Se describe con detalle el problema que se pretende resolver, se especifican los **requisitos** que el algoritmo debe cumplir. Además, se consideran factores éticos y legales, como la propiedad intelectual, se identifican y analizan riesgos potenciales, como la baja adopción y el renacimiento deficiente. Se establece un plan de trabajo sólido y se evalúa el presupuesto del proyecto.
- **Diseño de la solución:** Se propone un diseño de la solución basado en técnicas de aprendizaje profundo. Se describe el marco teórico que sustenta este diseño y se presentan las hipótesis y los objetivos del trabajo.
- **Desarrollo de la solución propuesta:** Describe el proceso del **desarrollo del modelo neuronal** final de aprendizaje profundo, destacando los desafíos técnicos que se han superado y las **decisiones clave** que se han tomado en el proceso.
- **Implantación:** Se detalla cómo se ha llevado a cabo la solución del modelo a partir de una interfaz de usuario (GUI) y cómo se han añadido funcionalidades a la propia aplicación.
- **Pruebas:** Se muestran los resultados de las pruebas realizadas para validar la efectividad del modelo.
- **Conclusiones:** Se presentan las conclusiones del proyecto, reflejando los logros y los errores a lo largo de su desarrollo.

1.6. Convenciones

La **APA** establece un conjunto de **normas y convenciones** para escribir, citar referencias y organizar documentos. A continuación, se proporcionará una **guía general** sobre cómo está **estructurado** el último apartado de este Trabajo Final de Grado, el **apartado de Referencias**. El **formato escrito** es según las **normas APA 7**:

1. **Autores:** Se empieza por los apellidos seguidos de las iniciales del nombre. En caso de varios autores, se separan por comas y se usa el símbolo “&” antes del último autor.
2. **Fecha de publicación:** A continuación, se pone la fecha de publicación entre paréntesis.
3. **Título:** El título del artículo se escribe a continuación, utilizando solo la primera letra mayúscula y el resto en minúsculas. El título del libro se escribe después en cursiva.
4. **URL:** Si el recurso es en línea.

Las **normas APA 7** son bastante detalladas y cubren muchos más aspectos que los resumidos en este apartado. Por lo que, si desconoce las normas APA, es recomendable que visite la **página web** [2].

2. Estado del arte

La **tecnología de visión por computador** ha experimentado avances significativos en la detección del lenguaje de signos en los últimos años. Los investigadores han desarrollado algoritmos y modelos de aprendizaje automático que pueden reconocer los gestos de las manos y otros movimientos asociados al lenguaje de signos.

El uso de **redes neuronales** es uno de los enfoques más comunes en el reconocimiento del lenguaje de signos. Estas redes están diseñadas para aprender patrones y características específicas en las imágenes, lo que les permite reconocer gestos y movimientos con las manos en tiempo real.

A continuación, se presentará la situación actual de la tecnología utilizada para el desarrollo de aplicaciones basadas en el reconocimiento del lenguaje de signos, así como algunos trabajos relevantes realizados en este campo.

2.1. Hardware

En este apartado, se realizará el estado del arte en el ámbito del entorno de programación relacionado con el desarrollo de algoritmos basados en el aprendizaje profundo para el reconocimiento del lenguaje de signos español. Se analizarán las **tecnologías y recursos** más utilizados en esta área.

2.1.1. Entornos en la nube

Los **entornos en la nube** para la ejecución de aplicaciones basadas en aprendizaje profundo proporcionan infraestructura hardware y software para desarrollar, entrenar y desplegar modelos de aprendizaje profundo.

Estos entornos proporcionan una serie de ventajas para el aprendizaje profundo permitiendo el acceso a **GPUs** (Graphics Processing Unit) y **TPUs** (Tensor Processing Unit) de alto rendimiento, que son esenciales para el entrenamiento de modelos de aprendizaje profundo. Además, son capaces de manejar grandes volúmenes de datos y ofrecen interfaces de usuario de fácil interpretación e integrados con servicios que facilitan la construcción de flujos de trabajo.

A continuación, se explicarán los **entornos de nube** que más se están utilizando en la **industria**.

Para más detalles acerca de Google Colab consulte [3], Amazon SageMaker [4], Microsoft Azure Machine Learning [5] e IBM Watson Studio [6] .

2.1.1.1. Google Colab

Colab se trata de un entorno gratuito basado en **Jupyter Notebook** que permite ejecutar **Python** y otros **lenguajes de programación (R)** directamente en tu navegador. Proporciona acceso gratuito a **GPUs** y es ampliamente utilizado para la experimentación en aprendizaje profundo debido a su facilidad de uso y a su integración en los servicios de Google.

El **servicio** es **gratis** para su **uso básico** y proporciona acceso a la GPU de manera gratuita. Además, Jupyter ofrece una interfaz sencilla de usar. Los **inconvenientes** que presenta Google Colab son las **cesiones temporales** (máximo de 12 horas) y sus **tiempos de inactividad** (aproximadamente 90 minutos)

2.1.1.2. Amazon SageMaker

Se trata de un s**ervicio AWS** que permite construir, entrenar y desplegar modelos de **machine learning** de manera rápida y fácil. **SageMaker** proporciona una serie de funciones para simplificar el proceso de entrenamiento distribuido y capacidades de inferencia en tiempo real.

Proporciona un entorno completamente gestionado para entrenar y desplegar modelos de machine learning y ofrece una amplia gama de herramientas y funcionalidades, incluyendo el **ajuste automático de modelos (AutoML)**. Los contras que ofrece Amazon SageMaker es el **coste económico**.

2.1.1.3. Microsoft Azure Machine Learning

La diferencia que tiene este servicio con respecto a los demás es que ofrece una **interfaz visual** para los usuarios sin experiencia en la codificación, así como **SDKs** para usuarios más avanzados.

Además, es capaz de integrarse con otras herramientas de **Microsoft**, como **PowerBI y Excel,** para el posterior análisis en los datos. Como aspecto negativo, está el **precio**.

2.1.1.4. IBM Watson Studio

Se trata de un entorno pensado para **científicos de datos**. Permite el desarrollo colaborativo de **modelos de machine learning** y proporciona acceso a una variedad de herramientas.

Es interesante esta opción, ya que proporciona una gama amplia de herramientas incluido **Jupyter** y ofrece una interfaz gráfica basada en arrastrar y soltar. Pero, su integración con servicios no-IBM puede ser costosa.

2.1.2. Hardware

Es una opción ideal para la implementación de aplicaciones relacionadas en el ámbito de la **Inteligencia Artificial**, ya que poseemos el control total con la capacidad de personalizar la configuración para satisfacer las necesidades de este trabajo. Además, una vez se haya adquirido el Hardware, no hay costes adicionales.

2.1.2.1. Módulos de la NVIDIA Jetson

Los módulos NVIDIA Jetson [7] son reconocidos por su impresionante capacidad para manejar tareas informáticas de IA con eficiencia. Además, la facilidad y versatilidad de la implementación son evidentes, todos ellos utilizando una pila de software común equipada con el kit de desarrollo

de software (SDK) Jetpack, que incluye un paquete de soporte de placa (BSP), un sistema operativo Linux y CUDA.

La **Jetson Nano** [8] es una **plataforma de inteligencia artificial** y un módulo de la Jetson que es accesible, potente y eficiente en el consumo de energía. Esta plataforma fue diseñada específicamente para permitir la **ejecución de aplicaciones modernas de AI y de alto rendimiento**, como el aprendizaje profundo, el procesamiento de imágenes y la robótica computacional, en un dispositivo pequeño, compacto y de bajo consumo.

Dicho módulo ofrece una GPU de 128 núcleos. Dicho recurso es suficiente para poder utilizar las librerías que Python nos ofrece para trabajar en desarrollos basados en Inteligencia Artificial. Se puede ejecutar Pandas, Numpy, TensorFlow y Keras. Ya que ofrece capacidad de cómputo más que suficiente para poder realizar los cálculos.

Figura 1. *NVIDIA Jetson Nano*

Existen otros **módulos**, como la NVIDIA Jetson TX2/TX2i, la NVIDIA Jetson Xavier NX, la NVIDIA Jetson AGX Xavier, la NVIDIA Jetson Orin NX o la NVIDIA Jetson AGX Orin. A partir de la **Tabla 1**, se puede ver una **comparativa con las características de cada tarjeta hasta la AGX Xavier** [9].

A través de la **Tabla 2**, se puede ver que el **módulo de la NVIDIA AGX Orin es el más potente** [10], ofrece una GPU de 2048 núcleos de CUDA con 32 GB de memoria RAM DDR5 y una frecuencia de reloj de 2GHz enfocado en conseguir una mejor eficiencia energética. El módulo en comparación a sus competidores ha dado un salto en términos de potencia, que la convierte en el **mejor módulo Jetson del mercado**, pero con un **precio elevado** para las funcionalidades que se le exigirá, es por ello, que **se descarta esta opción**. En cambio, la **NVIDIA Jetson Nano** parece la más razonable, ya que es la **placa más económica** y es **capaz de soportar los cálculos para el desarrollo del modelo neuronal desarrollado**.

	Jetson Nano	Serie Jetson TX2				Jetson Xavier NX	Serie Jetson AGX Xavier	
		TX2 NX	TX2 4 GB	TX2	TX2i		AGX Xavier	AGX Xavier Industrial
Rendimiento de IA	472 GFLOPS	1,33 TFLOPS			1.26 TFLOPS	21 PRINCIPALES	32 PRINCIPALES	30 TOPS
GPU	GPU NVIDIA Maxwell™ de 128 núcleos	GPU NVIDIA Pascal™ de 256 núcleos				GPU NVIDIA Volta™ de 384 núcleos con 48 núcleos Tensor	GPU NVIDIA Volta de 512 núcleos con 64 núcleos Tensor	
CPU	Procesador de cuatro núcleos ARM® Cortex®-A57 MPCore	CPU de doble núcleo Denver 2 de 64 bits y procesador de cuatro núcleos Arm Cortex-A57 MPCore				CPU de 64 bits NVIDIA Carmel Arm® de 6 núcleos v8.2 L2 de 6 MB + L3 de 4 MB	CPU de 64 bits NVIDIA Carmel Arm® de 8 núcleos v8.2 L2 de 8 MB + L3 de 4 MB	
Acelerador DL	-	-				2x NVDLA	2x NVDLA	2x NVDLA
Acelerador de visión	-	-				Procesador VLIW Vision de 7 zócalos	2 procesadores VLIW Vision de 7 zócalos	
Motor de clúster de seguridad	-	-				-	-	2 x Arm Cortex-R5 en lockstep
Memoria	LPDDR4 de 4 GB y 64 bits a 25,6 GB/s	LPDDR4 de 4 GB y 128 bits a 51,2 GB/s		LPDDR4 de 8 GB y 128 bits a 59,7 GB/s	LPDDR4 de 8 GB y 128 bits (Soporte ECC) a 51,2 GB/s	LPDDR4x de 8 GB y 128 bits a 59,7 GB/s	LPDDR4x de 32 GB y 256 bits a 136,5 GB/s	LPDDR4 de 32 GB y 256 bits (Soporte ECC) a 136,5 GB/s
Potencia *	5 W | 10 W	7,5 W | 15 W			10 W | 20 W	10 W | 15 W | 20 W	10 W | 15 W | 30 W	20W | 40W
Almacenamiento	eMMC 5.1 de 16 GB	eMMC 5.1 de 16 GB		eMMC 5.1 de 32 GB	eMMC 5.1 de 32 GB	eMMC 5.1 de 16 GB	eMMC 5.1 de 32 GB	eMMC 5.1 de 64 GB

Tabla 1. *Comparativa de módulos de NVIDIA Jetson hasta la AGX Xavier*

	NANO	TX2 NX	XAVIER NX*	ORIN NX 8 GB	ORIN NX 16 GB	AGX XAVIER**	AGX ORIN 32GB	AGX ORIN 64GB
GPU	128 Core Maxwell 0.5 TFLOPs (FP16)	256 Pascal cores 1.3 TFLOPS (FP16)	384 Core Volta 21 TOPs (INT8)	1024 Cores Ampere NVDLA V2 70 TOPS (INT8)	1024 Cores Ampere 2x NVDLA V2 100 TOPS (INT8)	512 Core Volta + NVDLA 10 TFLOPS (FP16) 32 TOPS (INT8)	1792 Cores Ampere 2x NVDLA V2 200 TOPS (INT8)	2048 Cores Ampere 2x NVDLA V2 275 TOPS (INT8)
CPU	4 core ARM A57	6 core Denver and A57 (2x) 2MB L2	6 core Carmel ARM V8 (3x) 2MB L2 + 4MB L3	6 cores A78 ARM V8 1.5MB L2 + 4MB L3	8 cores A78 ARM V8 2MB L2 + 4MB L3	8 core Carmel ARM V8 (4x) 2MB L2 + 4MB L3	8-core A78 ARM V8 2MB L2 + 4MB L3	12 core A78 ARM V8 3MB L2+ 6MB L3
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	4 GB 128-bit LPDDR4 51.2 GB/s	8GB or 16GB 128-bit LPDDR4x 51.2 GB/s	8 GB 128-bit LPDDR5 102.4 GB/S	16 GB 128-bit LPDDR5 102.4 GB/S	32GB or 64 GB 256-bit LPDDR4x 137 GB/s	32 GB 256-bit LPDDR5 204.8 GB/s	64 GB 256-bit LPDDR5 204.8 GB/s
Storage	16 GB eMMC	16 GB eMMC	16 GB eMMC	- (Supports external NVMe)	- (Supports external NVMe)	32 GB eMMC	64 GB eMMC	64 GB eMMC
Encode	4K @ 30 (H.265)	4K @ 60 (H.265)	2x 4K @ 30 (H.265)	1x 4K @ 60 (H.265)	1x 4K @ 60 (H.265)	4x 4K @ 60 (H.265)	1x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)
Decode	4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	6x 4K @ 60 (H.265)	2x 4K @ 60 (H.265)	3x 4K @ 60 (H.265)
Camera	12 (3x4 or 4x2) MIPI CSI-2 D-PHY 1.1 lanes (18 Gbps)	12 lanes (3x4 or 5x2) MIPI CSI-2 D-PHY 1.2 (30 Gbps)	12 lanes (3x4 or 6x2) MIPI CSI-2 D-PHY 1.2 (30 Gbps)	8 lanes MIPI CSI-2 D-PHY 1.2 (20 Gbps)	8 lanes MIPI CSI-2 D-PHY 1.2 (20 Gbps)	16 lanes MIPI CSI-2 | 8 lanes SLVS-EC D-PHY (40 Gbps) C-PHY (59 Gbps)	16 lanes MIPI CSI-2 D-PHY 2.1 (40 Gbps) C-PHY 2.0 (164 Gbps)	16 lanes MIPI CSI-2 D-PHY 2.1 (40 Gbps) C-PHY 2.0 (164 Gbps)
Mechanical	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin connector	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin edge connector	69.6mm x 45mm 260 pin edge connector	100mm x 87mm 699 pin connector	100mm x87 mm 699 pins connector	100mm x87 mm 699 pins connector
Software	JetPack SDK - Unified software release across all Jetson products							

Tabla 2. *Especificaciones de cada módulo de la NVIDIA Jetson*

2.1.2.2. Hardware dedicado

Las **tarjetas gráficas de NVIDIA** son conocidas por su **rendimiento en la renderización de gráficos de alta calidad para videojuegos**. Sin embargo, en los últimos años, NVIDIA ha ampliado su enfoque para tareas relacionadas con la **Inteligencia Artificial** incluyendo la aceleración de la computación paralela, a través de su **arquitectura de hardware CUDA y cuDNN** [11] NVIDIA Developer. (1 de mayo de 2022). *NVIDIA cuDNN*. NVIDIA. .

CUDA es una plataforma de computación paralela y un modelo de programación desarrollado por **NVIDIA** que permite a los desarrolladores utilizar las **GPU** para la computación de propósito general. Con **CUDA**, los desarrolladores pueden acelerar las aplicaciones basadas en aprendizaje profundo y realizar cálculos intensivos, al permitir que el software aproveche la gran cantidad de núcleos de procesamiento paralelo disponibles de la **GPU de NVIDIA**.

En referencia a **cuDNN** es una **biblioteca de software desarrollada por NVIDIA** para **acelerar las operaciones primitivas de redes neuronales profundas en las GPU**. CuDNN proporciona implementaciones altamente optimizadas de funciones primitivas, como la convolución y la

normalización, que son fundamentales para las redes neuronales convolucionales y otras arquitecturas de redes neuronales profundas.

Estas funciones optimizadas en cuDNN permite a los desarrolladores centrarse en el diseño y la implementación de modelos de redes neuronales sin tener que preocuparse por las optimizaciones de bajo nivel. Las **bibliotecas de aprendizaje profundo** utilizadas para este trabajo como **TensorFlow y PyTorch** utilizan **cuDNN para acelerar el entrenamiento y la inferencia en las GPUs de NVIDIA**.

Las **tarjetas gráficas de NVIDIA**, junto con la **plataforma CUDA** y la **biblioteca cuDNN**, proporcionan una **potente combinación de hardware y software** que permite acelerar las tareas de **Inteligencia Artificial**, desde el entrenamiento de modelos de aprendizaje profundo hasta la inferencia en tiempo real en aplicaciones de IA. A continuación, se explicará la **construcción de un equipo de alta gama para tareas de Inteligencia Artificial**, detallando la **importancia de cada componente**.

2.1.2.2.1. Construcción de un equipo

En caso de querer **construir un equipo propio de Inteligencia Artificial** [12] , es importante identificar el **Hardware relevante**. En este caso, la parte más importante es la **tarjeta gráfica**. Sin duda las partes como la **CPU** o la **memoria RAM** también son importantes, pero la GPU llega a conseguir tiempos de operaciones más rápidos que en CPU. Es por ello por lo que la **GPU cobra relevancia** en detrimento de otros elementos Hardware.

Lo aconsejable es **adquirir dos GPUs**, ya que los conjuntos de datos de **Deep Learning** suelen ser pesados, por lo que es importante poder paralelizar. Si se busca el **mejor rendimiento sin importar el aspecto económico**, se barajarían **dos GPUs RTX 4090** de **24GB VRAM y 16384 núcleos CUDA**. Para el **Aprendizaje Profundo**, se deberá de tener en cuenta la **memoria de la gráfica**. Cuanta más memoria, se podrá entrenar cantidades más grandes de datos a la vez (batch) por lo que se ahorrará tiempo.

En cuanto al **procesador**, si bien no es lo más importante, sí que se necesita tener una **CPU** lo suficientemente potente como para mover los datos y que no se creen cuellos de botella. Además, se puede preprocesar los datos correctamente en caso de que no se pueda hacer con la GPU. Es por ello, que lo aconsejable en cuestión de potencia es una **CPU intel I9 13900KS de 24 núcleos y 3.2 GHz.** Se escoge **procesador Intel** y no AMD, debido a que Intel tiene **mejores adaptaciones en el ámbito de Machine Learning,** y en este caso, se busca estabilidad. Es cierto que los procesadores AMD ofrecen más hilos, pero la elección del procesador es según las necesidades del desarrollador.

El papel de la memoria **RAM** es crucial, particularmente en tareas de procesamiento y preprocesamiento de datos. En ciertas ocasiones, la **GPU puede no ser capaz de manejar todo el procesamiento**, por lo que una **memoria RAM abundante** es esencial. El **aspecto más importante que considerar al seleccionar una RAM** para este Trabajo Final de Grado es su **capacidad**. Esto se debe a que los procesadores sólo admiten hasta cierta velocidad de transferencia de datos, creando un cuello de botella donde la RAM no puede entregar datos a la velocidad deseada. Aunque se podría aumentar la velocidad del procesador (overclocking) para soportar entregas de datos más rápidas, el incremento en la velocidad no variaría en más de unos

pocos segundos tras horas de entrenamiento. Por tanto, recomendaría una **memoria RAM abundante**, como **64 GB.**

El resto de **los componentes hardware**, se **adaptarán según la compatibilidad de los elementos previamente mencionados**. Se deberá buscar una **placa base compatible**, junto con un buen **disipador para la CPU**, un **disco duro SSD con gran almacenamiento** y una **Fuente de Alimentación con certificación**.

Todos estos elementos sirven para **construir un equipo profesional para ejecutar problemas de Inteligencia Artificial** en los que se requiera **capacidad de cómputo para realizar operaciones a gran escala**. En este aspecto, se ha fijado en la máxima potencia, pero debido al precio total de construir un equipo desde cero, se llega a la conclusión de que es preferible un **módulo de Jetson para el desarrollo de nuestro proyecto**.

2.2. Software

A continuación, se realizará el estado del arte en el ámbito del **software** relacionado con el desarrollo de algoritmos capaces de reconocer el Lenguaje de Signos Español. En este apartado se abordarán una amplia gama de aspectos y tecnologías software actuales en el desarrollo de la inteligencia artificial.

2.2.1. OpenCV

OpenCV [13] es una **biblioteca de código abierto** ampliamente utilizada en el campo de la visión por computador y el procesamiento de imágenes. Fue desarrollada originalmente por **Intel** y actualmente es mantenida por la **comunidad de código abierto**. OpenCV proporciona una amplia gama de funciones y algoritmos que permiten el procesamiento y análisis de imágenes y vídeos en tiempo real.

Está **escrita en C++** y cuenta con interfaces para los lenguajes de programación más populares, como **Python y Java**. Proporciona funciones para realizar operaciones básicas en imágenes, como filtrado, transformaciones geométricas, detección de bordes y segmentación. También incluye algoritmos más avanzados, como reconocimiento facial, seguimiento de objetos, calibración de cámaras y reconstrucción 3D.

Se utiliza en una amplia gama de aplicaciones, como sistemas de vigilancia, automatización industrial, realidad aumentada, robótica, reconocimiento de objetos y análisis de imágenes médicas. Su popularidad se debe a su facilidad de uso, rendimiento, versatilidad y su enfoque de código abierto, que permite a los desarrolladores acceder y modificar el código fuente según sus necesidades.

A continuación, se presenta un **ejemplo de detección de gestos con las manos utilizando OpenCV** [14] Ilango, G. (6 de abril de 2017). Hand Gesture Recognition using Python and OpenCV-Part 1. *Gogulilango*. . En este programa, se realiza el reconocimiento de una mano mediante la eliminación del fondo. Es importante destacar que se requiere un fondo plano y simple para su correcto funcionamiento. Además, la mano debe ubicarse dentro de un área específica en la cámara.

En la **Figura 2**, se puede ver la ejecución de un algoritmo capaz de detectar las manos mediante OpenCV.

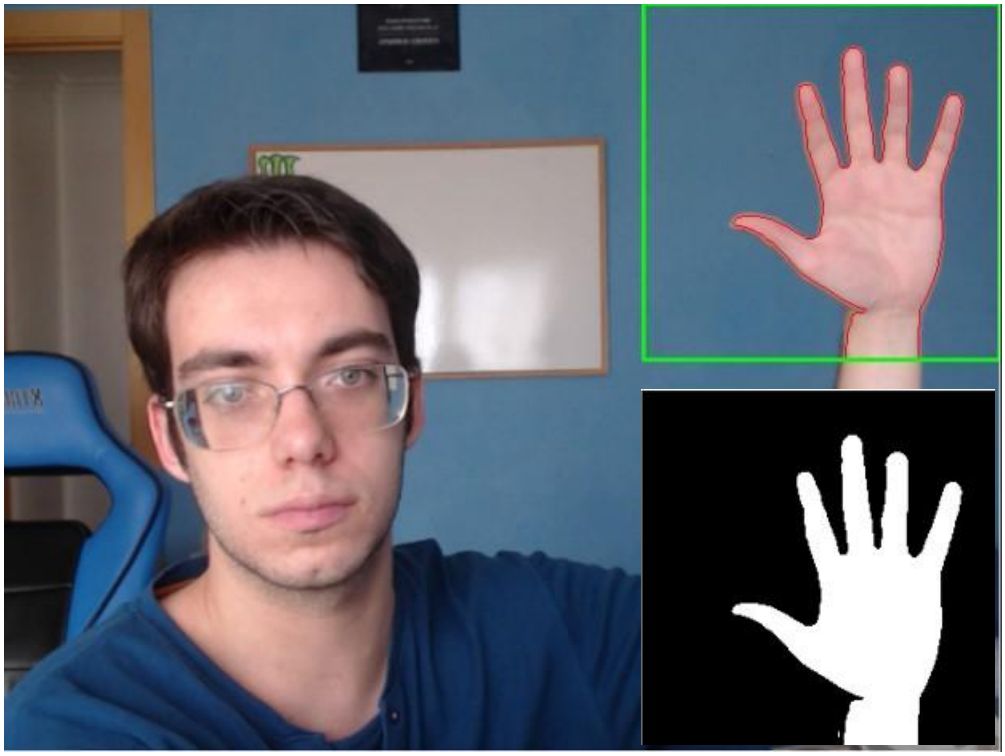

Figura 2. *Ejecución del algoritmo utilizando OpenCV*

Otra alternativa para el **reconocimiento de gestos mediante OpenCV**, el **programa** [15] es un ejemplo de implementación que se basa en la comparación de imágenes de gestos predefinidos con los gestos capturados por la cámara en tiempo real, aprovecha la utilización de **cv2.matchTemplate()** para realizar una comparación de plantillas, es decir, para encontrar la similitud entre una imagen de entrada y una plantilla de gesto específico.

Esta función toma tres argumentos: la imagen de entrada, la plantilla y un método de coincidencia. El método de coincidencia especifica cómo se calculará la similitud entre la imagen y la plantilla. En este caso, el método utilizado es **cv2.TM_CCOEFF_NORMED**, que calcula la correlación cruzada normalizada entre la imagen y la plantilla. Proporciona un valor numérico que indica cuán similar es la imagen a la plantilla. Un valor alto indica que hay similitud.

La función **cv2.matchTemplate()** desliza la plantilla sobre la imagen y compara cada región de la imagen con la plantilla, calculando el grado de similitud. Luego, devuelve la matriz con los resultados de la comparación en cada posición.

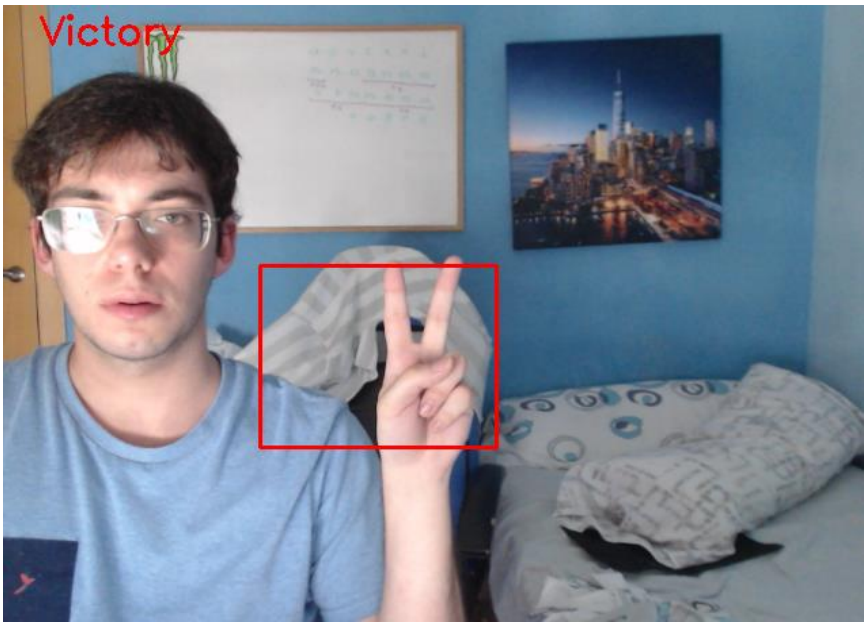

Figura 3. *Demostración a través de la función cv2.matchTemplate()*

A continuación, se mostrará una **solución basada en aprendizaje profundo para el reconocimiento del Lenguaje de Signos Español a través de OpenCV**, emulando la **funcionalidad** de **Microsoft Kinect mediante una webcam** [16] Heintz, B. (17 de diciembre de 2018). Training a neural Network to Detect Gestures with OpenCV in Python. *Towards Data Science*. . El **software** captura imágenes en tiempo real a través de la cámara web y procesa estas imágenes para detectar y reconocer los gestos. Para este reconocimiento, se utiliza un **modelo neuronal** que cuenta con **capas densas**. Luego, de acuerdo con los gestos identificados, se ejecutan acciones específicas que son integrables con **dispositivos de hogar inteligente**. Para visualizar el **resultado final de este programa,** se comparte el siguiente **enlace** [17] Heintz, B. [scgrinchy]. (18 de diciembre de 2018). *Gesture Recognition Demo with Python & OpenCV* [Archivo de Vídeo]. Youtube. .

2.2.2. TensorFlow

TensorFlow [18] es una **biblioteca para el aprendizaje automático** capaz de desarrollar modelos Machine Learning en JavaScript y usar Machine Learning directamente en el navegador o en Node.js.

TensorFlow proporciona múltiples modelos para una variedad de aplicaciones; tales como clasificaciones de imágenes, detección de objetos, segmentación corporal, detección de poses, etc.

Para la detección de manos, TensorFlow proporciona un modelo interesante **MediaPipe Handpose** [19] , se trata de una **biblioteca de código abierto desarrollada por Google** que utiliza la tecnología de detección y seguimiento de manos para estimar la pose de manos en tiempo real. Proporciona una interfaz fácil de usar para detectar y rastrear las manos en imágenes o secuencias de vídeo, lo que permite aplicaciones interactivas basadas en gestos en tiempo real.

La biblioteca utiliza un modelo de aprendizaje automático pre entrenado para detectar la presencia y la ubicación de las manos en una secuencia de video o imagen. Utiliza redes neuronales para identificar regiones de interés que contienen manos en diferentes posiciones y ángulos.

Una vez que se detectan las manos, **MediaPipe Handpose** realiza un seguimiento de las posiciones de las articulaciones de los dedos y la palma de la mano a medida que se mueven en el cuadro [20] . Utiliza algoritmos de seguimiento robustos para mantener un seguimiento preciso incluso cuando las manos se mueven rápidamente.

Utilizando la información de detección y seguimiento, se estima la pose completa de la mano, incluyendo la posición y orientación de la palma, así como las posiciones de las articulaciones de los dedos. Esto permite realizar un seguimiento preciso de los gestos y movimientos de la mano en tiempo real.

En la **Figura *4***, se puede ver la ejecución de la **utilización de MediaPipe Handpose** [21].

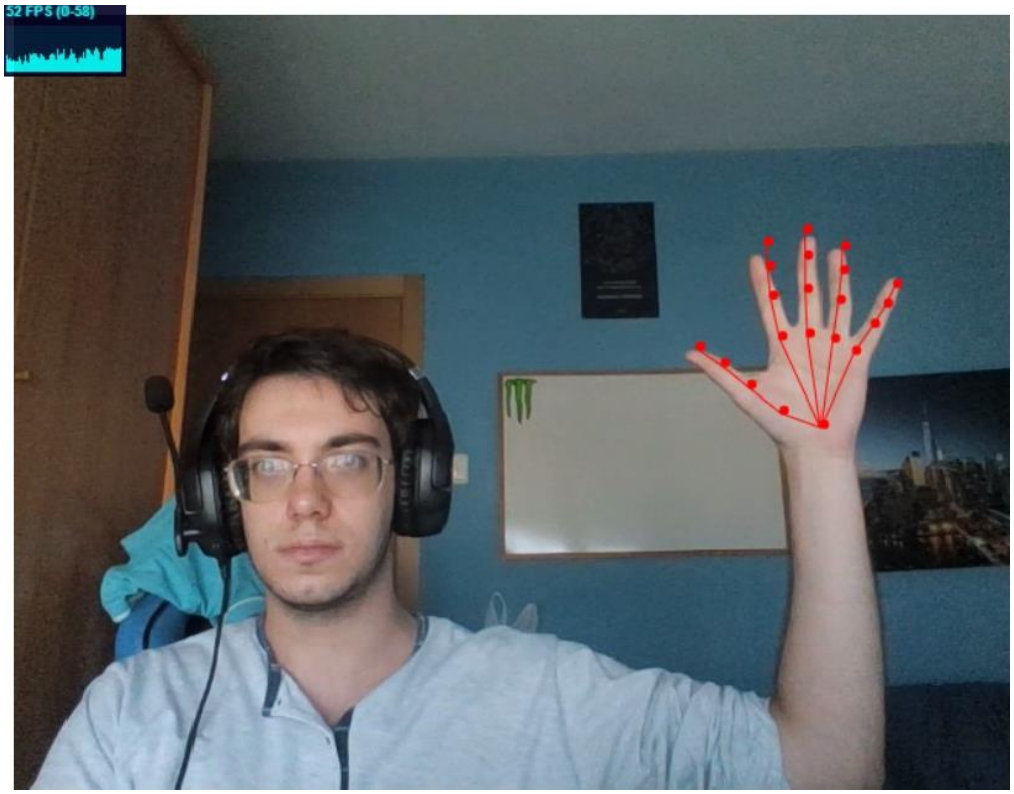

Figura 4. *Ejecución MediaPipe Handpose*

También existe una base de datos de reconocimiento de gestos de manos en una colección de imágenes casi infrarrojas divididas en diez gestos distintos [22] . A partir de este conjunto de datos, se desarrolló un algoritmo basado en redes neuronales convolucionales para construir un clasificador de gestos [23] Harrington, B. (20 de julio de 2018). Hand Gesture Recognition Database with CNN. *Kaggle*. .

2.2.3. PoseNet

PoseNet [24] es un **modelo de visión por computador** desarrollado por **Google** que puede estimar las poses humanas en tiempo real a través de imágenes y videos. Este modelo es capaz de

detectar la posición de una persona en una imagen a través de la identificación de puntos clave en el cuerpo.

Algunos puntos clave en el desarrollo de PoseNet ha sido la **detección de presencia de puntos clave en una imagen**, a través de la **Figura *5*,** se puede ver la detección que hace a través de 17 puntos clave.

PoseNet es una biblioteca para ejecutar estimaciones de posturas en tiempo real en el navegador mediante la biblioteca **TensorFlow**.

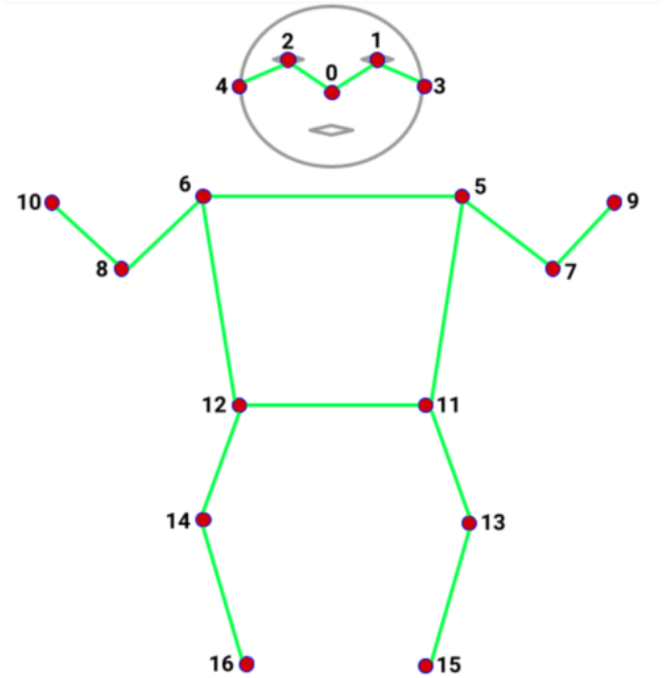

Figura 5. *Detección de los puntos clave a través de PoseNet*

Para cada punto clave detectado, PoseNet proporciona las coordenadas de su ubicación en una imagen y una puntuación de confianza que indica la probabilidad de que ese punto fue correctamente localizado.

Cabe mencionar que la estimación que realiza PoseNet, no difiere con respecto a la profundidad del objeto en un espacio tridimensional. Es decir, proporcionará las coordenadas (x, y), pero no la coordenada z.

A través de la **Figura *6*,** se puede ver una demostración de la utilización de **PoseNet** a través de una webcam [25] .

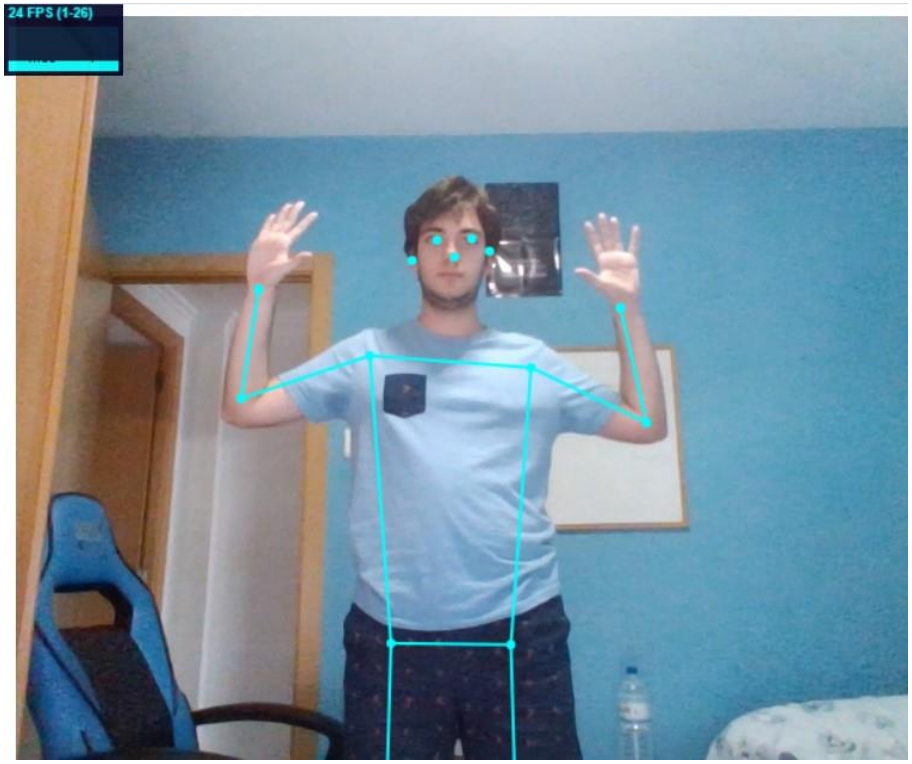

Figura 6. *Ejecución de PoseNet*

2.2.4. Pytorch

Pytorch [26] es una **biblioteca de aprendizaje automático** desarrollada por **Facebook**. Ha sido ampliamente utilizado en el ámbito de la inteligencia artificial, especialmente en el campo del **aprendizaje profundo**.

Posee varias características notables para aprovechar eficientemente la potencia de procesamiento de la **GPU**. La diferencia principal entre TensorFlow y Pytorch es en la utilización de grafos de computación dinámicos, lo que facilita la depuración y el diseño de modelos dinámicos, debido a que se puede modificar el grafo en tiempo de ejecución. Además, posee una **extensa librería** de funciones para tareas de visión por computador, para tareas de audio y para tareas de procesamiento de texto.

Respecto a la **utilidad de Pytorch** para el algoritmo desarrollado en este trabajo, es una opción para considerar, ya que es fácil de trabajar con **datos multidimensionales** como las imágenes procesadas.

2.2.5. Keras

Keras [27] es una **biblioteca de Python utilizada para desarrollar y experimentar con redes neuronales.** Ofrece una interfaz amigable y modular que facilita la construcción y modificación de los modelos de aprendizaje profundo.

Al ser modular, se permite a los usuarios construir redes neuronales aplicando capas y ajustándose según las necesidades, en este caso, se desarrollará para la visión por computador. Keras es muy

flexible y esta flexibilidad se extiende a la creación de funciones de pérdida personalizadas, lo que significa que se puede crear una red que se adapte a las necesidades según qué proyecto en el ámbito de la Inteligencia Artificial.

Además, ofrece una **forma estructurada de organizar las capas**. Existen dos tipos de modelos, los modelos secuenciales y los modelos funcionales. Los **modelos secuenciales** son apropiados para redes sencillas que tienen entrada y salida, mientras que los **modelos funcionales** son para redes más complejas que pueden tener múltiples entradas y salidas.

Para **compilar los modelos**, se necesitan las funciones de pérdida y optimizadores. Las **funciones de pérdida** miden qué tan bien está funcionando la red en la tarea. Los **optimizadores**, son los que ajustan los parámetros de red para minimizar la función de pérdida. Además, proporciona **métricas**, que son utilizadas para evaluar el rendimiento del modelo.

También, ofrece una serie de **funciones de activación** que determinan qué redes neuronales deben activarse en función de la entrada de datos. Además, ofrece **regularizadores**, se tratan de técnicas para evitar el sobreajuste del modelo, es decir, que aprenda demasiado bien los datos y no sean capaces de generalizar nuevos datos.

Keras está integrado en **TensorFlow 2.0**, lo que significa que se puede acceder desde TensorFlow.

2.3. Algoritmos para el reconocimiento del Lenguaje de Signos

Relacionado con las diferentes herramientas que se podrán utilizar para la detección de manos, se pueden desarrollar **algoritmos para reconocer el Lenguaje de Signos**. A continuación, se expondrán algunos **ejemplos de desarrollo**.

Un buen punto de partida es obtener un conjunto de datos que capture imágenes de personas realizando diferentes signos, donde posteriormente, se realizaría el etiquetado de cada imagen que represente el signo correspondiente.

En **esta página web** [28] Tecperson. (2 de agosto de 2017). Sign Language MNIST. *Kaggle*. , se muestra un conjunto de **datos MNIST de lenguaje de signos americano**, donde a partir de esa base de datos se realiza un algoritmo que utiliza redes neuronales convolucionales de la **librería de Keras** para realizar la predicción [29] .

También se han desarrollado modelos usando **OpenCV**, a través del siguiente repositorio [30] Kazi, A. (20 de mayo de 2023). Sign Language Recognition Using OpenCV. *Github.* , se puede ver el desarrollo de un algoritmo de reconocimiento del lenguaje de signos americano.

Google elaboró una competición para la detección y traducción del deletreo manual del lenguaje de signos americano (ASL) a texto [31] . Enviando un conjunto de datos que incluyen más de tres millones de caracteres escritos con los dedos, producidos por más de 100 personas sordas, y capturados mediante una cámara de teléfono, en una variedad de fondos y condiciones de iluminación.

Existen multitud de **startups** que desarrollan soluciones para el reconocimiento de lenguaje de signos. Por **ejemplo**, el **startup SignAll** [32] ofrece varios dispositivos en el mundo que permite

la comunicación entre una persona sorda y otra oyente a través de traducción automática de ASL (Lengua de Signos Americana).

En territorio europeo, el **proyecto Content4All** [33] , crea un humano virtual que replica de manera digital grabaciones de intérpretes reales que se expresan en lenguaje de signos.

Además, en territorio nacional, un grupo de **investigadores de la Facultad de Informática de la UPV/EHU** [34] ha desarrollado un sistema de reconocimiento de la lengua de signos argentina que utiliza puntos de referencia de la mano extraídos de vídeos con el fin de distinguir entre diferentes signos.

También, existen múltiples soluciones que han desarrollado en diferentes **Trabajos Finales de Grado** sobre esta situación, como ejemplo se puede ver [35] .

Existen numerosos ejemplos adicionales que podrían ser mencionados en relación con las soluciones existentes en el campo de la visión artificial para el reconocimiento del lenguaje de signos.

A partir de la **Figura *7***, se puede observar que, si se busca en el buscador "sign language recognition", la lista de resultados con diferentes soluciones es interminable [36] .

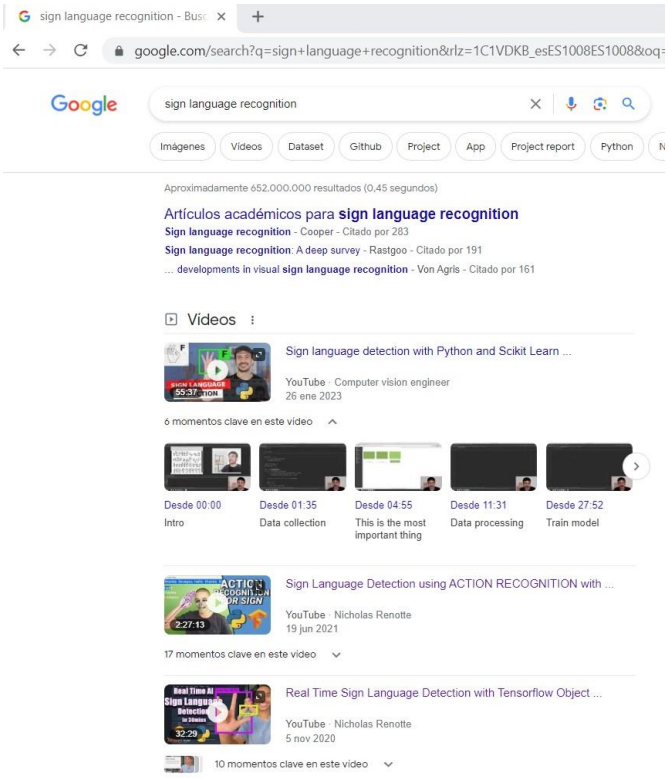

Figura 7. *Búsqueda de Sign Language Recognition*

2.4. Crítica al estado del arte

En la actualidad, el reconocimiento de signos ha logrado avances significativos. No obstante, persisten algunas **limitaciones** importantes que afectan la viabilidad de implementar estos algoritmos en entornos del mundo real. Uno de los principales desafíos es el **tiempo de respuesta**

prolongado al realizar predicciones durante el proceso de reconocimiento. Esta demora **impide una comunicación fluida y ágil**, lo cual es crítico en muchos casos. Es necesario abordar esta cuestión para mejorar la eficiencia y la aplicabilidad de los algoritmos de reconocimiento de signos en situaciones prácticas.

La causa principal se debe a que, generalmente, en los algoritmos de reconocimiento del lenguaje de signos se involucran diferentes técnicas de visión por computador y aprendizaje automático, que implica analizar y procesar grandes cantidades de datos. Lo que implica una **carga computacional significativa**.

Para lograr una **implementación exitosa** en entornos del mundo real, es necesario optimizar y mejorar los algoritmos para **reducir la carga computacional** y **acelerar los tiempos de respuesta** sin comprometer la precisión del algoritmo.

Además, se observa una carencia destacable de recursos específicamente diseñados para el Lenguaje de Signos Español que podrían ser explotados.

2.5. Propuesta

Este Trabajo Final de Grado tiene como **objetivo principal** el **reconocimiento de la dactilología del Lenguaje de Signos Español utilizando tecnologías de visión por computador y técnicas de aprendizaje profundo**. El conjunto del Trabajo Final abarca desde la recolección de datos hasta el entrenamiento de un modelo de reconocimiento de signos y la implementación de una interfaz interactiva.

El **espacio de conocimiento** que va a llenar es el **campo de la traducción y comunicación para personas con discapacidad auditiva que utilicen el Lenguaje de Signos Español**. En caso de que tenga éxito, el algoritmo desarrollado podrá potenciar la accesibilidad y la inclusión al permitir la comunicación entre personas que utilizan el lenguaje de signos y aquellos que no lo conocen.

Lo que **diferencia este trabajo de algunas de las soluciones existentes es su enfoque en el lenguaje de signos español**, específicamente, lo que permite adaptarse a las particularidades propias del idioma.

3. Análisis del problema

El **problema principal** que aborda el algoritmo basado en aprendizaje profundo para el reconocimiento del lenguaje de signos realizado para este Trabajo Final de Grado es la **comunicación efectiva entre personas con discapacidad auditiva que utilizan el lenguaje de señas español y aquellas que no lo conocen**. Y es que la barrera del idioma puede dificultar la interacción y limitar las oportunidades de inclusión y participación social para las personas con discapacidad auditiva. Además, ofrece una oportunidad de innovación al desarrollar un sistema de reconocimiento de señas del lenguaje de signos español utilizando técnicas de visión por computador y aprendizaje automático. Al mejorar la comunicación entre las personas que utilizan el lenguaje de signos y las que no lo conocen, se promueve la inclusión y se facilita el acceso a la información y los servicios para las personas con discapacidad auditiva.

Existen una serie de **requisitos** que el **algoritmo** debe cumplir:

- Se requiere un modelo de aprendizaje profundo entrenado a partir de redes neuronales para que el algoritmo reconozca y clasifique los signos del lenguaje español.
- El sistema debe de ser capaz de procesar la secuencia de signos y generar la traducción correspondiente en texto o voz.
- Se debe implementar una interfaz de usuario intuitiva y accesible que permita la interacción con el sistema.
- El algoritmo debe ser capaz de funcionar en tiempo real.
- Se requiere un **conjunto de imágenes dactilológicas** del lenguaje de signos español para el entrenamiento del modelo.
- El sistema debe tener una alta precisión en el reconocimiento de signos y una baja tasa de errores.

3.1. Análisis del marco legal y ético

El **marco legal** es de obligado cumplimiento por todo profesional y debe ser considerado desde diversos aspectos.

3.1.1. Propiedad intelectual

El proyecto desarrollado para el trabajo final de grado se desarrollará como proyecto de **código abierto** [37] Opensource. (6 de junio de 2023). *New developments at Opensource.com* , lo que significa que el código fuente estará disponible de manera **gratuita** y se podrá acceder y distribuir de manera abierta por parte de la comunidad de desarrolladores e interesados.

Al ser un **proyecto de código abierto**, se fomentará la colaboración y participación de la comunidad, permitiendo que cualquier persona pueda contribuir con mejoras, correcciones de errores, nuevas características y necesidades específicas. Ofreciendo de esta manera una transparencia, adaptabilidad y accesibilidad para brindar una solución mejorada para el reconocimiento del lenguaje de signos.

3.1.2. Ética

El entrenamiento del modelo de reconocimiento del lenguaje de signos requiere conjuntos de datos amplios y representativos que reflejen la **diversidad de los usuarios**. Para enfrentar este desafío, se ha adoptado un enfoque basado en el aprendizaje profundo utilizando **técnicas de transfer learning** para hacer uso de una **red pre entrenada (VGG19)**. Este enfoque permite evitar sesgos y garantizar la inclusión de una amplia variedad de usuarios. Al eliminar estos sesgos, se reduce el riesgo de discriminación y se promueve la equidad en el reconocimiento e interpretación del lenguaje de signos. Abordar este dilema ético es de vital importancia para asegurar que el modelo sea justo e inclusivo en su funcionamiento.

3.2. Análisis de riesgos

A continuación, se analizarán los **riesgos** que puedan suponer la implantación del desarrollo de una solución basada en el aprendizaje profundo para el reconocimiento del lenguaje de signos español.

- **Riesgo de aceptación**:
  - **Tipo de riesgo**: Baja adopción del sistema por parte de los usuarios objetivo.
  - **Impacto**: En caso de que el algoritmo de reconocimiento del lenguaje de signos no sea ampliamente adoptado por los usuarios, supondría una limitación en su impacto y utilidad en la comunicación del lenguaje de signos. Lo que afectaría negativamente la eficacia y viabilidad del trabajo final de grado.
  - **Medidas para reducir el impacto**:
    - Investigar y comprender las necesidades y preferencias de los usuarios objetivo.
    - Realización de pruebas durante el desarrollo para garantizar que el sistema cumpla con las expectativas.
- **Riesgo de rendimiento**:
  - **Tipo de riesgo**: Bajo rendimiento o tiempos de respuesta lentos del proyecto.
  - **Impacto**: Si el sistema tiene un rendimiento deficiente, los usuarios pueden experimentar retrasos en la detección y reconocimiento del lenguaje de signos, lo que afecta negativamente la eficiencia y experiencia del usuario.
  - **Medidas para reducir el impacto**:
    - Realización de **pruebas exhaustivas de rendimiento y optimización** para garantizar una detección y reconocimiento más rápido y preciso.
    - Utilización de técnicas de procesamiento de imágenes eficientes.

Es importante destacar que estas medidas anteriormente dichas no eliminan por completo los riesgos, pero ayudan a mitigar su impacto y a garantizar un entorno satisfactorio para los usuarios.

3.3. Identificación y análisis de soluciones posibles.

Al analizar los requisitos del Trabajo Final de Grado, se ha elaborado un conjunto de **herramientas alternativas** que abordan el problema del reconocimiento del lenguaje de signos. A continuación, se presentarán algunas de ellas analizando sus **pros y contras**, y se establecerá un **criterio de selección** para determinar la solución adecuada.

- **OpenCV:**
  - **Pros:** Es una biblioteca ampliamente utilizada y cuenta con una amplia gama de funcionalidades para el procesamiento de imágenes y visión por computador. Tiene soporte para diversos algoritmos de detección y seguimiento de objetos.
  - **Contras:** No está específicamente diseñado para el reconocimiento del lenguaje de signos. Se requerirá una implementación personalizada y entrenamiento adicional para adaptarse a los requisitos específicos.

- **Tensorflow:**
  - **Pros:** Es un marco de aprendizaje automático de código abierto ampliamente utilizado. Ofrece herramientas para entrenar y desplegar modelos de aprendizaje automático, lo que podría ser útil para el reconocimiento de gestos.
  - **Contras:** Requiere una curva de aprendizaje significativa para utilizar eficazmente. Además, el desarrollo de modelos personalizados puede ser complejo y requiere conjuntos de datos grandes y etiquetados.

- **MediaPipe HandPose:**
  - **Pros:** Es una biblioteca de Google que permite el seguimiento de las manos en tiempo real. Proporciona un rendimiento rápido. Es útil para detectar y seguir los gestos de las manos en el lenguaje de signos.
  - **Contras:** Se centra en el seguimiento de las manos y no en la interpretación de los gestos del lenguaje de signos. Requiere una adaptación y entrenamiento adicional para cumplir con los requisitos.

- **PyTorch:**
  - **Pros:** Es un popular marco de aprendizaje automático de código abierto que proporciona herramientas y funcionalidades para entrenar y desplegar modelos de aprendizaje automático. Tiene una comunidad activa y una amplia documentación.
  - **Contras:** Se requiere un conocimiento profundo de aprendizaje automático y su rendimiento depende de la complejidad del modelo.

- **PoseNet:**
  - **Pros:** Es una biblioteca desarrollada por Google que permite la detección de poses humanas en tiempo real utilizando modelos de aprendizaje automático. Puede ser útil para detectar y seguir las poses de las manos en el lenguaje de signos.
  - **Contras:** Se centra en la detección de poses y no en la interpretación de los gestos en el lenguaje de signos. Lo que requerirá personalización y entrenamiento adicional para adaptarse a los requisitos del trabajo final de grado.

- **Keras:**
  - **Pros:** Es una biblioteca de aprendizaje profundo de alto nivel que se ejecuta sobre Tensorflow. Proporciona una interfaz fácil de usar y abstrae gran parte de la complejidad del desarrollo de modelos de aprendizaje automático.
  - **Contras:** Al ser una biblioteca de alto nivel, existen limitaciones en términos de personalización y flexibilidad en la implementación de modelos de reconocimiento de signos altamente especializados. Además, puede requerir conocimientos

adicionales para integrar con otras bibliotecas necesarias para el procesamiento de imágenes y el seguimiento de manos.

Para seleccionar la solución adecuada con lo que se trabajará, se deben tener en cuenta diferentes aspectos como la capacidad de detección y seguimiento de las manos, la interpretación de los gestos, la facilidad de implementación, el rendimiento y eficiencia en términos de velocidad y en el cumplimiento de los requisitos del desarrollo del algoritmo.

Basándose en estos criterios, la **solución** que parece más **adecuada para el desarrollo del algoritmo** es **Tensorflow y Keras**, con la necesidad de **importar regularizadores y capas**. El algoritmo utilizará **técnicas de transfer learning** para usar un modelo **pre entrenado VGG19**, junto con **técnicas de procesamiento de imágenes**. En este caso, se hará uso de la **técnica de data augmentation** en los datos para poder adquirir más imágenes y de esta forma crear un modelo neuronal más eficiente.

3.4. Solución propuesta

En este Trabajo Final de Grado se tratará de desarrollar una **solución** basada en el **aprendizaje profundo para el reconocimiento de la dactilología del Lenguaje de Signos Español**. En esta sección, se presentará la solución elaborada por distintas **fases**.

1. **Análisis de requisitos**: En esta fase, se analizarán e identificarán los requisitos funcionales del Trabajo Final de Grado para su posterior desarrollo.
2. **Preparación del entorno de desarrollo**: Se estudiarán las diversas opciones y se buscarán las mejores tecnologías capaces de poder desarrollar una solución basada en el aprendizaje profundo para el reconocimiento del lenguaje de signos de manera amena, buscando la mayor facilidad posible.
3. **Recopilación de datos**: En este proceso se recopilarán los datos para entrenar el modelo de reconocimiento del lenguaje de signos. Los datos son imágenes realizando todos los signos dactilológicos de la lengua española.
4. **Preprocesamiento de imágenes**: Se llevan a cabo operaciones de generación de imágenes que serán utilizadas en etapas posteriores para el entrenamiento del modelo. Esta estrategia es de gran utilidad en este proyecto, dada la necesidad de trabajar con un gran conjunto de datos, que excede la capacidad de carga en la memoria de la Jetson Nano. Adicionalmente, se implementan transformaciones a las imágenes para enriquecer el conjunto de entrenamiento. Este procedimiento es conocido como **Aumento de Datos** (Data Augmentation), una técnica de Inteligencia Artificial para el procesamiento de imágenes.
5. **Entrenamiento del modelo**: Se entrenará un modelo de aprendizaje profundo para reconocer los signos. Para el entrenamiento del modelo, se utilizará aprendizaje por transferencia, se trata de una técnica eficaz en la visión por computador. La idea es utilizar el modelo pre entrenado VGG19 y realizar un ajuste fino en él, ajustando sus pesos para adaptarse al problema de clasificación de imágenes.
6. **Implementación de la aplicación**: Se desarrollará la aplicación de reconocimiento del lenguaje de signos. Lo que implica la creación de la interfaz de usuario y la integración del modelo entrenado para realizar la detección en tiempo real utilizando como medio una webcam.

7. **Pruebas y validación**: En esta fase, se realizarán pruebas para validar el rendimiento de la aplicación. Se evaluará la precisión y el rendimiento de la detección y del reconocimiento.
8. **Optimización**: Si se identifican posibles mejoras durante la fase de prueba, se realizarán optimizaciones y ajustes en la aplicación para mejorar la precisión y rendimiento.
9. **Redacción del informe**: Una vez completadas todas las fases anteriores, se redactará el informe final del Trabajo Final de Grado.

3.5. Plan de Trabajo

En este apartado, se realizará una estimación de **esfuerzo del desarrollo completo para una solución basada en aprendizaje profundo para el reconocimiento de lenguaje de signos**, se puede dividir todo el proceso en las siguientes **fases**:

1. **Comprensión y análisis de los requisitos**: En esta fase, se deben revisar en detalle los requisitos funcionales del Trabajo Final de Grado. Esto implica analizar los programas relacionados existentes, identificar las funcionalidades necesarias y comprender lo que se ha de hacer. El proceso puede requerir aproximadamente 20 horas, ya que, a lo largo del desarrollo de este trabajo, se han ido modificando los objetivos. Para alguien experimentado en la materia, se intuye que puede requerir 6 horas.
2. **Preparación del entorno de desarrollo**: Se debe configurar el entorno de desarrollo a partir del hardware escogido. Además de la instalación y configuración de las bibliotecas y dependencias necesarias, así como la configuración de la webcam. Esta fase puede requerir 4 horas aproximadamente.
3. **Recopilación de imágenes**: Se recopilarán las imágenes para entrenar el modelo. Se deberá de realizar cada gesto de la dactilología de la lengua española y capturar un gran número de imágenes clasificándolos en su etiqueta correspondiente. Esta fase puede durar aproximadamente 60 horas o 70 horas, es difícil de estimar, ya que en algunas ocasiones se han producido errores debido a que el signo está mal ejecutado por diferentes factores que se han ido solucionando a partir del desarrollo del Trabajo Final de Grado.
4. **Preprocesamiento de imágenes**: En esta fase, se utiliza la técnica de aumento de datos, que consiste en la realización de un generador de imágenes para crear un gran conjunto de datos. Estas transformaciones realizadas a partir de las imágenes recopiladas anteriormente se basan en transformaciones aleatorias. La duración de este proceso es difícil de cuantificar, pero se puede estimar unas 10 horas. Para un profesional en el ámbito de la inteligencia artificial, es posible que le requiera menos tiempo.
5. **Entrenamiento del modelo de reconocimiento del lenguaje de signos**: Implica la utilización de un aprendizaje por transferencia para poder optimizar el modelo neuronal lo máximo posible. Para ello, se utiliza el modelo VGG19 y se realiza un ajuste fino entrenado a partir de la técnica de aumento de datos anteriormente empleada. De esta manera, se crea el modelo neuronal a partir de una amplia gama de imágenes. El tiempo requerido varía mucho en función de la complejidad, número de capas, transfer learning y del tamaño del conjunto de datos. La elaboración de esta fase tuvo una duración de 50 horas aproximadamente.
6. **Desarrollo de la aplicación de reconocimiento en tiempo real**: Se implementa una construcción de la interfaz de usuario. Esto implica implementar el modelo entrenado

para que reconozca al usuario y pueda lanzar una predicción. Esta fase tardó 30 horas en realizarse.

7. **Pruebas**: Después de completar la implantación, es necesario realizar pruebas exhaustivas para verificar el correcto funcionamiento del proyecto de final de grado. Esto incluye probar cada una de las señas de signos de la dactilología del Lenguaje de Signos Español, verificar la precisión del reconocimiento y solucionar cualquier problema encontrado. En esta fase, se necesitaron 5 horas de trabajo.
8. **Optimización**: A lo largo del desarrollo del proyecto final y tras realizar las pruebas necesarias, en muchas ocasiones se ha necesitado optimizar el rendimiento de la aplicación para mejorar la experiencia del usuario, lo que incluye mejoras en el código, revisión del modelo, aplicación de diferentes técnicas de preprocesado y corrección a la hora de recopilar los datos.
9. **Redacción del proyecto**: Creación de documentación técnica explicativa para la entrega y presentación ante un jurado, se presenta como Trabajo Final de Grado junto a la entrega del proyecto. La duración total es de 120 horas.

Cabe mencionar que se ha realizado un seguimiento constante del avance del proyecto, ajustando el plan según sea necesario y aprendiendo a lo largo de las desviaciones que se han ido produciendo.

3.6. Presupuesto

El principal **costo asociado** ha sido una **tarjeta NVIDIA Jetson Nano** por valor de 234,90 € [38] necesario para entrenar el modelo. Esto significa que se requirió de recursos computacionales para llevar a cabo el proceso de entrenamiento de manera fluida.

Por otro lado, los programas de software utilizados son de código abierto, disponibles gratuitamente.

4. Diseño de la solución

En este apartado, se llevará a cabo la **explicación de la solución al problema del reconocimiento del lenguaje de signos**. Se hará en dos niveles de detalle, explicando la **arquitectura que tendrá el sistema** y **se expondrá su diseño**. Además, se explicará la **tecnología utilizada** para este Trabajo Final de Grado.

4.1. Arquitectura del sistema

Se puede dividir la **arquitectura del proyecto** en **tres programas** independientes pero que a partir de su ejecución se comunican entre sí para la elaboración del proyecto:

1. **Captura y almacenamiento de datos**: Este es el programa responsable de la recopilación de imágenes y su almacenamiento. Es el primer paso y la base para la creación del modelo de aprendizaje profundo. Este paso es de especial importancia, ya que el modelo trabajará con los datos que se hayan elaborado. Es por ello por lo que se debe prestar especial atención.
2. **Preprocesamiento de datos y Entrenamiento del modelo**: A partir del almacenamiento de imágenes realizado del anterior script, se debe realizar un generador de imágenes. Los generadores de datos elaboran grandes conjuntos de datos para posteriormente entrenarlos. En este script, se realiza esta técnica para aplicar transformaciones en las imágenes y, de esta forma, conseguir un gran conjunto de datos que serán utilizados para entrenar al modelo neuronal y lograr un algoritmo óptimo en la clasificación de imágenes en tiempo real. Posteriormente, para entrenar el conjunto de datos que se ha creado, aplicamos transfer learning y se utiliza un modelo pre entrenado (VGG19) de ImageNet. Se aplica ajuste fino para evitar el sobreajuste (overfitting) y se guarda el modelo para su posterior implementación en la API.
3. **Implementación de la API**: Este programa actúa como el punto de salida de la construcción del modelo neuronal. Se basa en una aplicación diseñada a partir de ventanas, con dos funcionalidades, la de traducir y la de reconocer. Este programa presentaría los resultados finales del desarrollo del proyecto.

La **arquitectura de software** escogida es una **arquitectura en capas,** ya que cada ejecución del programa proporciona una capa de funcionalidad que construye la capa anterior.

A continuación, se mostrará el **diagrama de bloques** del Trabajo Final de Grado, a través de las **Figura *8.***

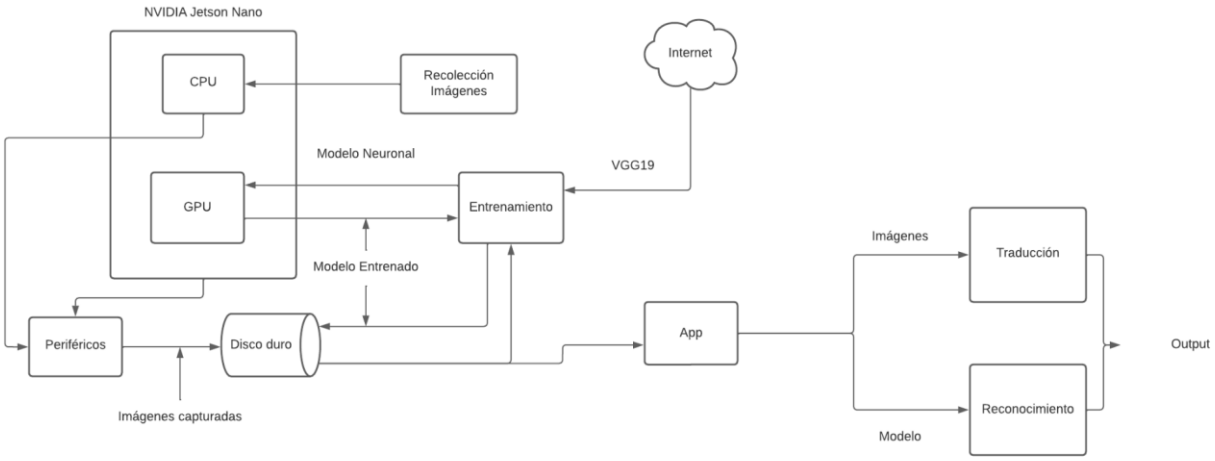

Figura 8. *Diagrama de Bloques*

Finalmente, se destacarán los **componentes esenciales de la arquitectura**: el **entorno** utilizado para la ejecución del proyecto es una **tarjeta NVIDIA Jetson Nano** y el **lenguaje de programación** utilizado es **Python** [39] . Para la **captura de imágenes** se ha utilizado una **webcam** y para el **entrenamiento del modelo** se ha usado la **biblioteca TensorFlow**. No obstante, para la **implementación GUI** del proyecto se ha usado **TKinter** [40].

Cada uno de los componentes interactúa en un paso específico del flujo general del sistema.

4.2. Diseño detallado

En este apartado, se definirá el **diseño del software**, donde se proporcionará la explicación de cómo se han realizado las clases y de cómo se relacionan entre sí.

4.2.1. Clase recoleccionDatos.py

Es la **primera clase** que ejecutará el programa. Su función es **capturar y guardar las imágenes de la webcam**. A continuación, se describe paso a paso lo que hace el programa.

Inicialmente, el programa **importa las bibliotecas necesarias para su correcta ejecución**. En este escenario, **'os' y 'sys'** son bibliotecas de **Python** encargadas de proporcionar diversas funciones de gestión de archivos, mientras que **'cv2'** es la biblioteca de **OpenCV** que ofrece funciones destinadas al procesamiento de imágenes.

Después de la importación de las bibliotecas, el programa procede a la **creación de una ventana y la definición del directorio donde se almacenará el conjunto de imágenes** que serán capturadas por la webcam. A continuación, **crea un diccionario** llamado **'count'**, el cual utiliza todas las letras del alfabeto como llaves y asigna a cada una de ellas un valor inicial de 0. Este diccionario tiene como objetivo llevar un registro de las imágenes guardadas correspondientes a cada letra del alfabeto.

Una vez definidos estos elementos, el programa procede a **establecer la función 'show_camera()'**. Esta función se encarga de abrir el dispositivo de la webcam. Tras verificar

que la webcam se ha abierto correctamente, la función entra en un ciclo indefinido, donde realiza una serie de acciones: primero, lee un fotograma de la cámara y lo muestra en una nueva ventana. En esta imagen, se dibuja un rectángulo blanco cuyo contenido se muestra en una segunda ventana denominada "ROI" (Region of Interest).

A continuación, el programa **espera a que el usuario presione una tecla**. Si el usuario presiona la tecla ESC, se sale del ciclo; pero si presiona una letra del alfabeto, el programa guarda la ROI en un archivo cuyo nombre incluye el número de veces que se ha pulsado dicha tecla. Así, cada signo realizado es almacenado de tal manera que, si se pulsa una tecla, la imagen correspondiente se guarda en una carpeta etiquetada con la tecla pulsada, proporcionando así un conjunto de imágenes etiquetadas.

Finalmente, en caso de que la cámara no se haya abierto correctamente, la función imprimirá un error. Cabe destacar que este programa resulta extremadamente útil para la recolección de datos de imágenes destinados a entrenar un modelo de aprendizaje profundo. Con estas imágenes, el programa se prepara para una fase posterior de entrenamiento.

4.2.2. Clase entrenamiento.py

La **segunda clase** [41] trata de **desarrollar el clasificador de imágenes**, específicamente para clasificar el lenguaje de signos español. El proyecto utiliza la **red neuronal convolucional VGG19** [42] Simonyan, K., & Zisserman, A. (10 de abril de 2015). *VGG16 and VGG19.* que ha sido pre entrenada en el dataset de ImageNet.

El **código** comienza **importando las librerías necesarias**. En este caso, se importará la librería **TensorFlow** para la construcción, entrenamiento y validación del modelo de clasificación de imágenes. Además, se incluye la biblioteca **Keras**, que está dentro de TensorFlow. También se importa **numpy**, para convertir la imagen en una matriz que puede ser utilizada por el modelo neuronal y **matplotlib**, para trazar la precisión y pérdida del modelo durante el entrenamiento y validación, lo que permite visualizar el rendimiento del modelo.

A continuación, se **inicializa un objeto 'ImageDataGenerator'**, que es una herramienta útil para generar lotes de imágenes con datos aumentados en tiempo real. Los datos aumentados son versiones modificadas de los datos de entrada, que ayudan a evitar el sobreajuste del modelo. El generador se configura para hacer cosas como reescalar las imágenes, aplicar un rango de zoom, cambiar la altura y el ancho, y cambiar el brillo.

Se **crean dos generadores de imágenes**, **'train_generator'** y **'valid_generator',** que cargarán las imágenes desde el directorio y las preprocesarán a partir de **'ImageDataGenerator'**. Estas imágenes se utilizarán para entrenar y validar el modelo.

Luego, se **carga el modelo VGG19 pre entrenado**. Para este modelo, las primeras seis capas se congelan, lo que significa que no serán entrenadas en el proceso de entrenamiento. Se agregan más capas que serán entrenadas.

A continuación, el modelo se compila con la pérdida de entropía cruzada categórica, el optimizador SGD y se mide la accuracy. Es difícil afirmar con certeza por qué SGD funciona mejor que Adam. La **elección del optimizador** puede depender de múltiples factores. Es posible que, aunque Adam tienda a converger más rápido que SGD, **Adam** puede ser más susceptible a

la adaptación excesiva produciendo overfitting. Esto se debería a que el conjunto de datos es pequeño. En contraste, **SGD** a menudo es más robusto y proporciona un mejor rendimiento de generalización en este caso.

Se **definen dos callbacks para el proceso de entrenamiento**: **'ModelCheckpoint'** para guardar el mejor modelo en función de la precisión de validación y **'ReduceLROnPlateau',** para reducir la tasa de aprendizaje cuando la pérdida de validación deja de mejorar. Se optó por utilizar ReduceLROnPlateau, ya que se adapta mejor para el optimizador SGD.

Luego, el **modelo se entrena** con datos generados por los generadores de imágenes.

Finalmente, el **modelo se guarda** en formato **json y h5** y se grafican la precisión y la pérdida del modelo durante el entrenamiento.

4.2.3. Clase app.py

Este es un **script** para una **aplicación de reconocimiento y traducción del lenguaje de signos español**.

Este programa utiliza **diversas bibliotecas** como **cv2** para procesar imágenes, **TensorFlow** para cargar el modelo de aprendizaje automático y **tkinter** para crear la interfaz de usuario.

La **aplicación** tiene **varias partes**:

1. La **función 'load_model'** carga el modelo de aprendizaje automático guardado en el proyecto.
2. Para **construir la interfaz de usuario de Tkinter**, se utilizan varias clases, incluyendo la página de inicio ('StartPage'), una página de información ('Info'), una página para traducir texto a lenguaje de signos español ('Pepe') y una página para reconocer lenguaje de signos desde la webcam ('Prueba').
3. La f**unción 'show_camera'** abre la cámara y utiliza el modelo de aprendizaje profundo para reconocer el lenguaje de signos español dactilológico. Los gestos detectados a partir del modelo se muestran en la interfaz de usuario.
4. La **función word2faces** es para la traducción de texto a lenguaje de signos, convierte un texto introducido por el usuario a lenguaje de signos español. Cada letra del texto se asocia con una imagen que representa el gesto correspondiente en lenguaje de signos. La función 'procesadoImagenes' se utiliza para mostrarlas en la interfaz de usuario.
5. La **función 'text_to_speech'** convierte el texto reconocido a voz a partir de la biblioteca pyttsx3.

Finalmente, el script termina con la **ejecución de la aplicación** realizada a través de **Tkinter**.

4.3. Tecnología utilizada

La **NVIDIA Jetson Nano** es el **hardware** utilizado para el desarrollo del Trabajo Final de Grado. Dicha tarjeta ofrece una GPU con 128 núcleos. Este recurso se puede **utilizar para aplicaciones**

de IA. Se puede ejecutar Pandas, Numpy, TensorFlow y Keras en una placa NVIDIA Jetson Nano. Además, se hará cargo de los cálculos que se deban realizar para la ejecución del programa.

El **lenguaje de programación escogido para el desarrollo del algoritmo** ha sido **Python**. Se trata de uno de los lenguajes de programación más utilizados en el aprendizaje automático y en la inteligencia artificial, debido a su facilidad de aprendizaje, versatilidad y la robustez de sus bibliotecas. Su sintaxis sencilla y legible hace que sea fácil de aprender y usar. Además, puede utilizarse para una amplia variedad de tareas, incluyendo el procesamiento de datos, la visualización de datos y el aprendizaje profundo. Otra característica de Python es la **amplia gama de bibliotecas y marcos de trabajo para la visión por computador** que ofrece, como **TensorFlow, Keras, Numpy, Pandas y Matplotlib**, todas ellas utilizadas para el desarrollo de este Trabajo Final de Grado, lo que ha facilitado el desarrollo de la solución al reconocimiento del lenguaje de signos español basado en aprendizaje profundo. Además, Python es ampliamente utilizado en la comunidad científica, por lo que hay una amplia comunidad de usuarios que proporcionan apoyo y asistencia. Todas estas características han hecho que Python haya sido la opción más atractiva para el desarrollo del proyecto.

En cuanto al **entorno de programación**, se ha escogido el **Visual Studio Code** [43] . Se ha escogido el entorno de Visual Studio Code por motivos idiosincrásicos, dicho editor de código fuente ofrece un autocompletado inteligente en Python basado en tipos de variables, definiciones de funciones y módulos importantes. Además, viene con capacidades de depuración integradas, lo que significa que no hay necesidad de cambiar de herramienta para rastrear los errores.

Existen extensiones en Python que Visual Studio Code ofrece, que son capaces de añadir soporte para el lenguaje de programación. Además de la facilidad que ha dado trabajar con múltiples archivos al mismo tiempo. Para la programación en Python, cabe destacar su resaltado de sintaxis de código. También soporta plegado de código, lo que permite ocultar secciones de código.

La elección de un editor de código es un asunto sumamente personal y subjetivo, razón por la cual se encuentran diversos desarrolladores empleando distintos editores de código en función de sus preferencias y necesidades. Personalmente, Visual Studio Code ha sido el editor que he utilizado de manera más frecuente durante mi trayectoria académica. Este compromiso con VS Code se ha cimentado sobre las ventajas y características únicas que el software ofrece, aspectos que se han mencionado anteriormente y que son la esencia de mi preferencia por este particular editor de texto.

5. Desarrollo de la solución propuesta

En este apartado se presenta el **desarrollo y evolución** de nuestro proyecto desde la concepción de la idea inicial hasta la implantación final. A lo largo del recorrido, se expondrán las **dificultades y desafíos** encontrados y cómo se han superado, además se detallarán las **decisiones clave** que han determinado la trayectoria y el resultado final.

5.1. Problemas y dificultades del proyecto

En principio, la idea para el desarrollo de este Trabajo Final de Grado ha sido optar por el diseño de una **solución basada en el aprendizaje profundo** [44] , ya que ha demostrado ser una herramienta eficaz en cuanto al desarrollo de tareas de reconocimiento de patrones complejos, como es en este caso.

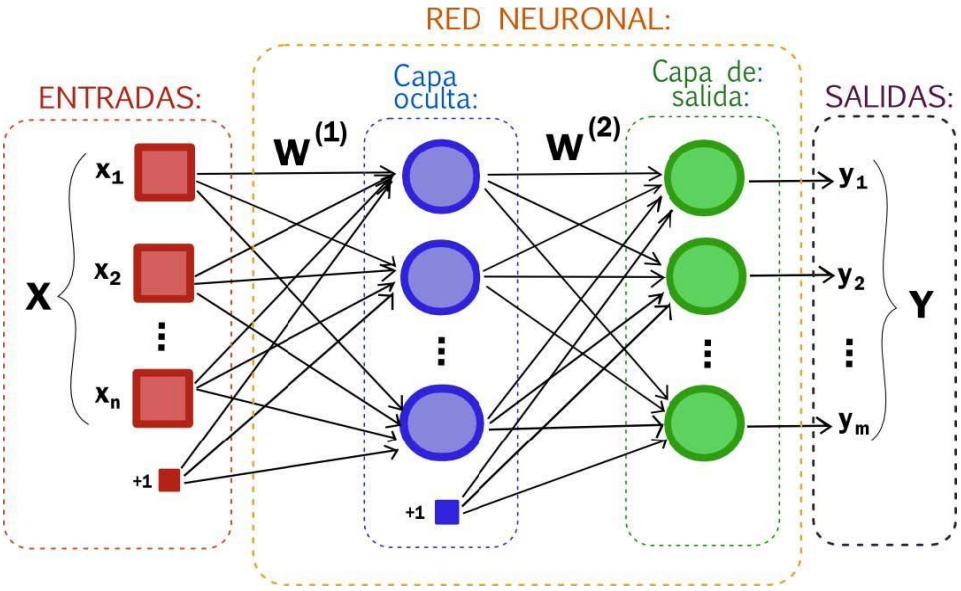

Figura 9. *Estructura de una red neuronal*

La solución que barajar fue optar por un **desarrollo de redes neuronales convolucionales** [45], estas redes son particularmente eficaces para el procesamiento de imágenes. En este caso, estas redes eran utilizadas para reconocer los gestos del lenguaje de signos a partir de las imágenes estáticas. Las imágenes de las manos realizando los signos sería el input, y el output sería la letra correspondiente.

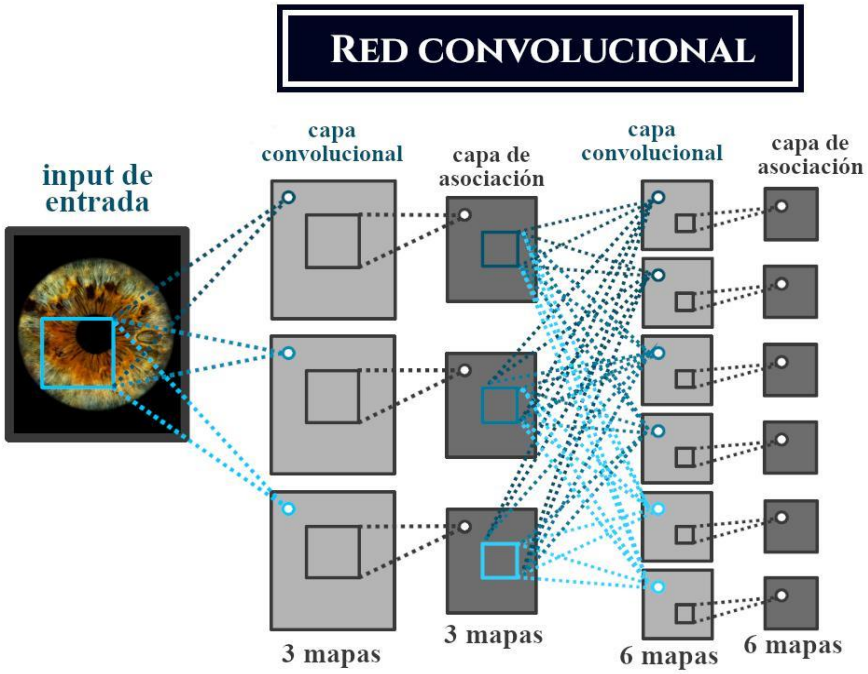

Figura 10. *Ejemplo de Red Neuronal Convolucional (CNN)*

Para el **desarrollo de datos** en el que trabajaría el modelo neuronal, se hizo desde cero. Debido a la falta de contenido en la red, se decidió partir de una **base de datos personalizada**, donde el **ejemplo de las imágenes** se puede ver a través de la **Figura *11***, dicha imagen captura la letra C en lenguaje de signos español. Debido al gran alfabeto que tiene la lengua española para el lenguaje de signos, se decidió reducir el conjunto de datos a la dactilología del Lenguaje de Signos Español. Por tanto, el **desarrollo de la aplicación** será el r**econocimiento del vocablo del lenguaje de signos español**, el alfabeto está compuesto de 26 letras.

Para la captura de imágenes, se desarrolló un script que cumpliera esta función y se capturaron un total de 2600 imágenes. Debido a la falta de imágenes, se aplicó técnicas de **Data Augmentation** [46] Daoust, M. (26 de enero de 2022). *Aumento de datos.* TensorFlow. : rotaciones, desplazamientos y cambios de escala. Para que el modelo mejorase de manera generalizada.

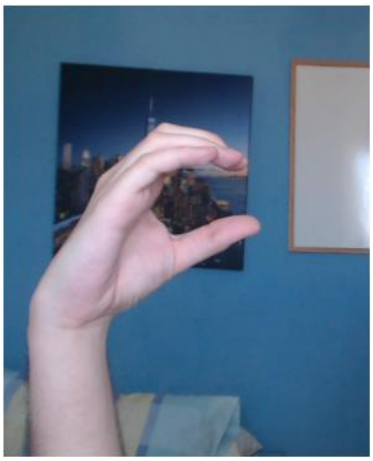

Figura 11. *Imágenes capturadas por primera vez*

El **modelo de la red neuronal** se trata de un **modelo secuencial**, cuya entrada son imágenes dimensionadas en 200x200.

Tras la ejecución del algoritmo a partir del modelo, hubo sobreajuste (overfitting), por lo que al testearlo se detectaron múltiples errores. La prueba realizada fue a partir de 5 etiquetas, se decidió probar con las vocales, ya que son las letras más utilizadas en el ámbito del lenguaje de signos español. Se analizó el error e intuimos que podría haber sido por la falta de regularización. La regularización agrega un costo a la función de pérdida del modelo que es proporcional a los pesos de los parámetros, esto ayuda a prevenir el overfitting reduciendo su complejidad.

También se utilizó **técnicas Dropout**, es otra forma de regularizar el algoritmo que funciona desactivando aleatoriamente una proporción de las neuronas durante el entrenamiento. Esto ayuda a asegurar que el modelo no se vuelva demasiado dependiente de ninguna neurona individual y fomenta una distribución de las características.

La **estructura del modelo neuronal** quedaría de esta forma. Se puede ver a través de la **Figura *12***. Cabe destacar que el modelo se modifica de muchas maneras, pero la estructura final en el que se obtuvo el mejor resultado es la que se proporciona.

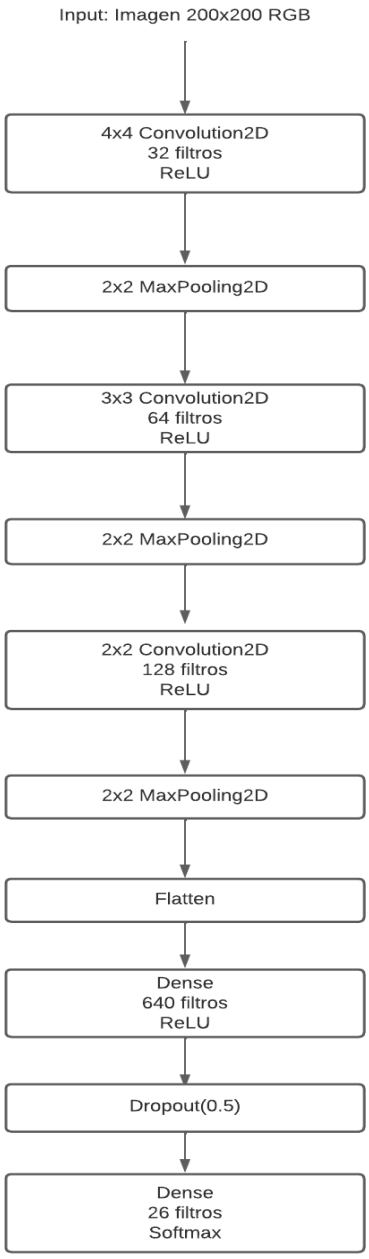

Figura 12. *Estructura de la Red Neuronal Convolucional diseñada*

Al entrenar el modelo, se consigue eliminar el overfitting, pero no se consigue funcionar en tiempo real, el modelo predice erróneamente el conjunto de imágenes que le llega en tiempo real. Por lo que, al analizar los datos, se intuye que es debido a que la captura de imágenes está mal realizada, a partir de la **Figura *11***, se puede ver que la **captura está mal tomada,** hay un cuadro de fondo, que podría producir ruido a la imagen que dificulta a que el algoritmo pueda realizar su función correctamente, por lo que se decide rehacer los datos.

Los **nuevos datos** se pueden ver a través de la **Figura *13*.** En esta captura de imágenes, se **elimina cualquier objeto que no tenga relación con la mano**, por lo que se corrige el posible problema que pudiera haber.

Figura 13. *Captura de imágenes correctamente realizada*

Tras volver a tomar las imágenes con la misma cantidad de fotos (2600 imágenes) y tras aplicar las técnicas de **Data Augmentation**, se prueba el modelo con la misma estructura. Pero vuelven a fallar a la hora de ejecutarlo en tiempo real. El algoritmo acierta con imágenes sueltas en las que previamente no ha visto, pero falla a la hora de ser ejecutado en tiempo real. Por lo que, se decide utilizar la **técnica de Transfer Learning** a través de **EfficientNet** junto a la creación del modelo. En la **Figura *14***, se puede ver la **estructura** creada del **modelo EfficientNet** [47] .

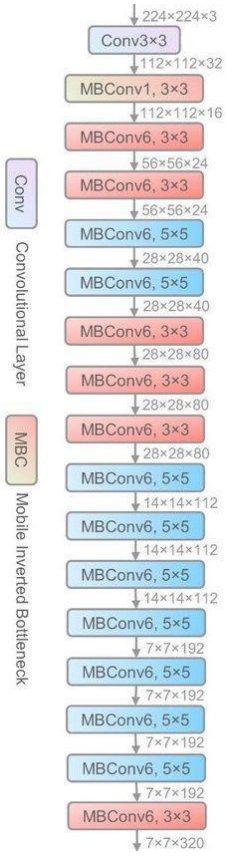

Figura 14. *Estructura del modelo EfficientNet*

El **transfer learning** [48] es una **técnica en la que se utiliza un modelo pre-entrenado** (en este caso **EfficientNet**) en un gran conjunto de datos como punto de partida. Este modelo ya ha aprendido muchas características generales y específicas, y se puede afinar para el problema del reconocimiento del lenguaje de signos español con menos datos de los que se necesitaría.

El motivo por el cual se decidió utilizar el transfer learning, es debido a las pocas imágenes de nuestro conjunto de datos. Este algoritmo fue probado, pero tampoco se logró solucionar la problemática en tiempo real.

Se intuye que la problemática del algoritmo para que no funcione es debido a la falta de imágenes, por lo que se decide dar otro enfoque al desarrollo del algoritmo, aprovechando la **librería MediaPipe** para el desarrollo de este. A través de la **Figura *15***, se puede ver la **estructura de la librería**.

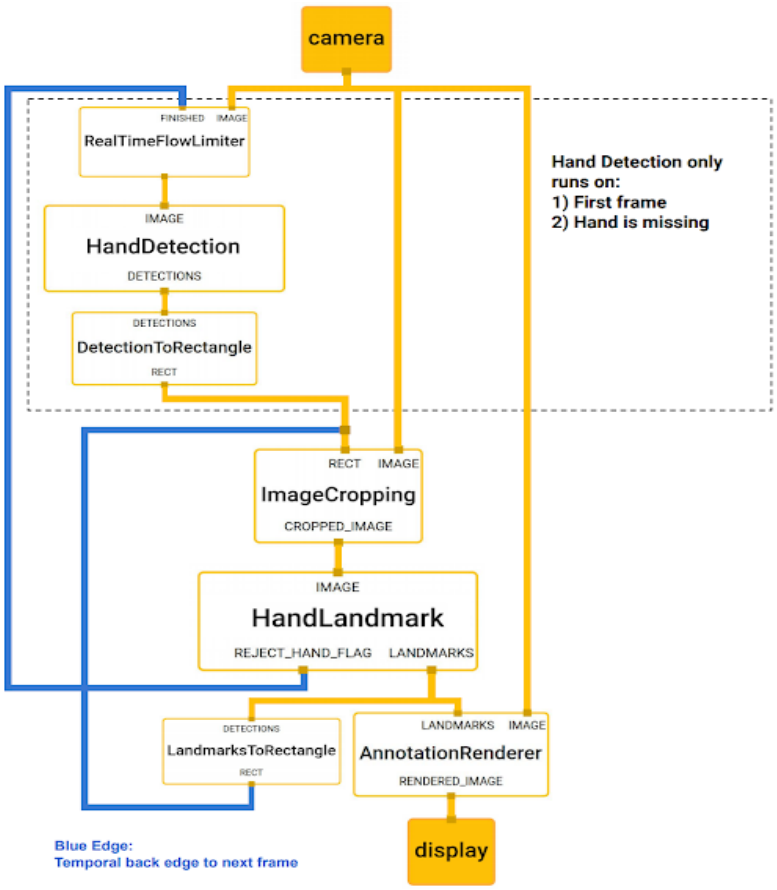

Figura 15. *Estructura de MediaPipe*

La decisión que se ha tomado para **solucionar la problemática** ha sido utilizar la **biblioteca MediaPipe**, es una **biblioteca que ofrece varios flujos de trabajo de machine learning preconstruidos**. Esto incluye soluciones para el seguimiento de manos. Los modelos en MediaPipe están optimizados para su uso en tiempo real en una variedad de plataformas, lo que los hace ideales para aplicaciones de procesamiento de imágenes en tiempo real. Ofrece una solución de seguimiento de manos capaz de **detectar y rastrear 21 puntos de referencia en 3D de una mano.** Estas características que aporta la librería hacen que se decida usar para la solución de este proyecto.

El enfoque realizado para el proyecto es que en lugar de que algoritmo sea entrenado por las imágenes, se entrene a partir de los **puntos de referencia** extraídos del conjunto de datos. Para hacer que el algoritmo funcione, se toman los datos desde distinta distancia y se extraen sus puntos clave de coordenadas, haciendo que el algoritmo reconozca el signo a partir de los puntos.

Para ello, se realiza de nuevo la captación de imágenes desde diferentes distancias marcadas. De esta manera, se obtiene un conjunto de imágenes preparado y basado en las imágenes de la Figura 5, en el que resalta la mano proporcionada.

A partir de estas imágenes se decide desarrollar un nuevo conjunto de datos a partir del procesado de las imágenes, lo que se intenta obtener es que a partir de las imágenes se extraigan diferentes posiciones de las coordenadas de los puntos clave. De esta manera, tenemos un array de datos con el conjunto de coordenadas.

Finalmente se desarrolla un nuevo modelo basado en **Redes Neuronales Recurrentes (RNN)**, en particular, **Long Short-Memory (LSTM)**, para aprender de las secuencias de movimientos. A partir de la **Figura *16***, se puede ver el funcionamiento de una Red Neuronal Recurrente (RNN) [49] IBM. (16 de marzo de 2023). *¿Qué son las redes neuronales recurrentes?* . Estos modelos son particularmente buenos para aprender patrones en secuencia de datos y funcionan bien en este contexto. La **arquitectura del modelo neuronal** final implementado para el desarrollo de una solución basada en el reconocimiento del Lenguaje de Signos Español dactilológico, se puede ver en la **Figura *17***.

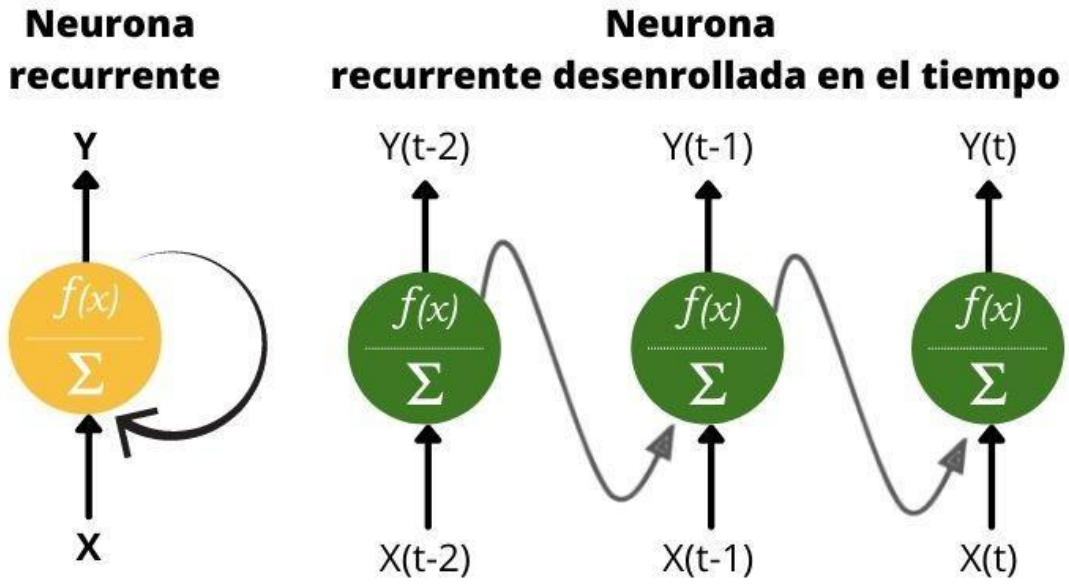

Figura 16. *Red Neuronal Recurrente (RNN)*

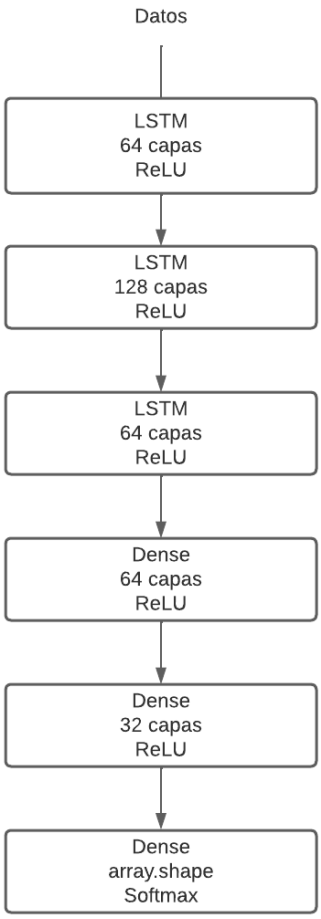

Figura 17. *Arquitectura del modelo neuronal recurrente*

Una vez que el modelo final ha sido almacenado y confirmado su correcto funcionamiento en tiempo real, se procede a realizar pruebas y se comprueba que el modelo no generaliza bien.

Para ilustrar la **funcionalidad de la aplicación que se ha desarrollado**, se ha preparado un **video demostrativo** [50] . En este material, se puede apreciar claramente cómo funciona nuestra aplicación en un escenario real.

El algoritmo funciona, pero no es capaz de cumplir con los objetivos marcados. No es generalizado.

Desarrollar una solución de aprendizaje profundo para un problema complejo como es el reconocimiento del lenguaje de signos, es una tarea desafiante en el que involucra muchos pasos y la superación de varios obstáculos. A continuación, se explicará el **desarrollo del algoritmo** en el que se ha logrado hacer que funcione y los diferentes **cambios que se han ido produciendo**, partiendo de la base de los distintos errores que se han cometido.

5.2. Solución Final

Para abordar el desafío, se adoptó una alternativa que excluye el uso de la biblioteca Mediapipe y la extracción de puntos clave. En su lugar, **se diseñó un modelo de red neuronal basado en aprendizaje profundo**, recurriendo nuevamente a la estrategia de transferencia de aprendizaje. Sin embargo, en lugar de aprovechar EfficientNet, se decidió por **VGG19**.

Con el objetivo de implementar una solución para el reconocimiento del lenguaje de signos basada en aprendizaje profundo, se importaron las bibliotecas necesarias, incluyendo TensorFlow, Matplotlib, PIL, regularizers, layers, optimizers, callbacks y preprocessing.

Antes de entrenar el modelo, es imperativo establecer un **generador de imágenes. Keras** proporciona la clase **'ImageDataGenerator'** para este fin, que permite generar lotes de datos de imágenes con aumento de datos en tiempo real para optimizar la eficacia del modelo.

Esta clase sirve para normalizar los valores de los píxeles, haciendo que la red neuronal sea más eficiente durante el entrenamiento. Se establece un corte de desplazamiento a lo largo de los ejes X o Y para generar nuevas imágenes a partir de las existentes. Además, se introduce un rango de zoom aleatorio y un movimiento aleatorio tanto en la dirección horizontal como vertical. Esto permite al modelo aprender a reconocer imágenes que no están centradas. Asimismo, se divide el conjunto de datos en grupos de entrenamiento y validación. El mejor rendimiento se logró al asignar un 20% de los datos para validación. Se ajusta aleatoriamente el brillo de las imágenes y se establece que no se deben voltear horizontal o verticalmente, lo cual es crucial para el lenguaje de señas español.

Estas **técnicas** implementadas en la clase **'ImageDataGenerator'** son formas de realizar el **aumento de datos (Data Augmentation)**, permitiendo un mejor entrenamiento del modelo. Al generar variaciones de las imágenes existentes, el modelo se vuelve más robusto y puede generalizar mejor a nuevas imágenes no vistas anteriormente.

Dentro de **'ImageDataGenerator'**, la **función 'flow_from_directory'** puede ser utilizada para cargar y preprocesar imágenes desde el directorio para el entrenamiento del modelo. Esta función permite alimentar los datos de entrenamiento y validación.

Durante el uso de esta función, se especifica que las etiquetas serán codificadas en formato one-hot, esto se debe a que se está realizando una clasificación multiclase y el formato 'categorical' es el adecuado. Las imágenes se seleccionan al azar y se redimensionan a un tamaño específico, en nuestro caso, (56, 56). Las imágenes se cargan en formato RGB y se especifica que son datos de entrenamiento, además del número de imágenes que se enviarán al modelo en cada paso de entrenamiento.

Para la **validación**, el proceso es similar, pero con la especificación de que son **datos de prueba.**

Una vez completado el procesamiento de los datos, se procede al **entrenamiento del modelo**. Para ello, se recurre al **modelo preentrenado VGG19**, un modelo neuronal convolucional profundo ideado por el Visual Geometry Group de la Universidad de Oxford, que goza de amplio uso en tareas de clasificación y detección de imágenes. La **Figura *18*** ofrece un **esquema** de la **estructura de VGG19**.

De este modelo, se explotará principalmente la **base convolucional**, descartando otras capas para ajustar el modelo a nuestras necesidades. Los pesos del modelo se heredan de **ImageNet**, un conjunto de datos extenso que incluye millones de imágenes de diversas clases, proporcionando un punto de partida sólido para extraer características útiles de las imágenes capturadas.

¿Por qué se eligió **VGG19 en lugar de EfficientNet** para este proyecto? Aunque **EfficientNet** generalmente presenta una eficiencia superior y una gran capacidad de cálculo, la selección de **VGG19** podría estar vinculada a la **compatibilidad de los datos**. ImageNet contiene datos más simples, mientras que EfficientNet suele ser más adecuado para conjuntos de datos complejos o de naturaleza diferente.

En el **proceso de configuración**, se **congelan diversas capas del modelo VGG19**. Esto se hace en el contexto del aprendizaje por transferencia, ya que las primeras capas de la red neuronal convolucional aprenden características de bajo nivel y generales, como bordes, texturas y colores, útiles para tareas de clasificación. Congelar estas capas permite **conservar estas características aprendidas y evitar el sobreajuste.**

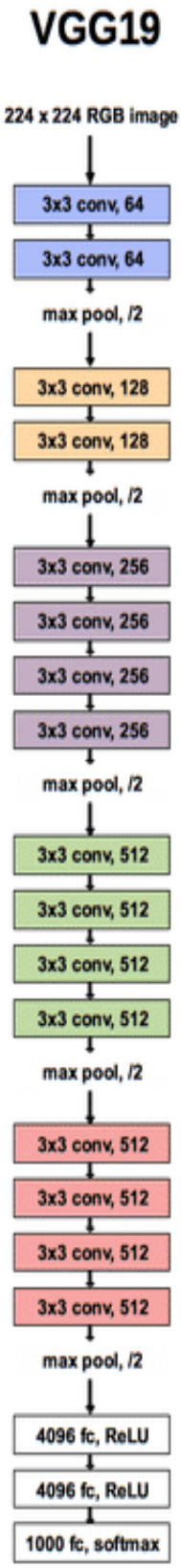

Figura 18. *Arquitectura del modelo VGG19*

Se **construye un modelo secuencial,** aprovechando la simplicidad de la API proporcionada por Keras para crear modelos de aprendizaje profundo con arquitecturas lineales. El **modelo**

preentrenado VGG19 se añade al **modelo secuencial** como una capa adicional, aprovechando todas sus capas y pesos.

Se incluyen también una **capa Flatten** para aplanar la entrada, convirtiendo las imágenes multidimensionales en unidimensionales; **dos capas Dropout** para desactivar un 25% de las neuronas durante cada época de entrenamiento, previniendo el sobreajuste; una **capa Dense** con 64 neuronas y función de activación ReLU, que ha demostrado ofrecer mejores resultados; y una última **capa Dense** con 26 neuronas, equivalente al número de clases en el problema, utilizando la función de activación softmax para obtener una distribución de probabilidad sobre las clases. A través de la **Figura *19***, se puede ver la **arquitectura final del modelo**.

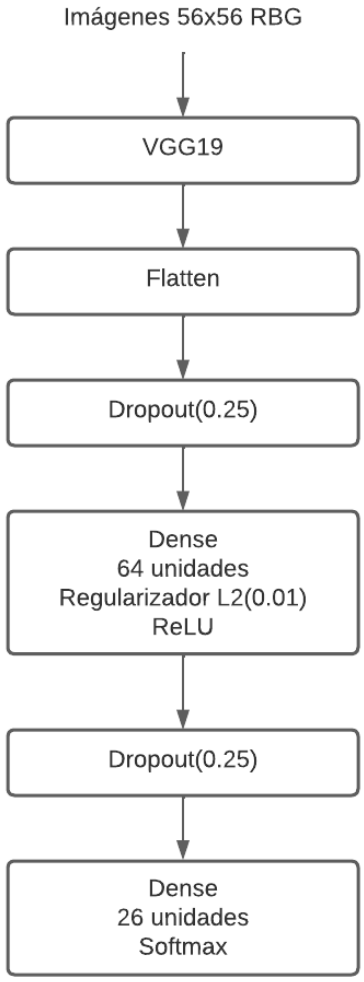

Figura 19. *Arquitectura del modelo neuronal con VGG19*

Tras diseñar la arquitectura del modelo neuronal, se prepara el modelo para su **entrenamiento con el método 'compile'**, definiendo la función de pérdida, el optimizador y las métricas. En este caso, se opta por la función de pérdida "categorical_crossentropy" para un problema de clasificación multiclase, el optimizador SGD y la métrica 'accuracy'.

Además, se añaden **callbacks para guardar los mejores resultados**, evitando el sobreajuste. Se definen dos callbacks: '**ModelCheckpoint**', para guardar el modelo después de cada época y '**ReduceLROnPlateau**', para reducir la tasa de aprendizaje cuando la métrica ha dejado de mejorar.

Finalmente, se **entrena el modelo con los datos de entrenamiento y validación**, determinando el número de veces que el algoritmo de aprendizaje recorrerá todo el conjunto de datos de entrenamiento. Posteriormente, se guarda la arquitectura del modelo y sus pesos en un **archivo HDF5**.

Tras probar el rendimiento del modelo, se puede afirmar que este cumple con los requisitos establecidos previamente, proporcionando **resultados satisfactorios** [51].

6. Implementación

A fin de desarrollar un **modelo neuronal para el reconocimiento del lenguaje de signos español**, la intención es implementar una **Interfaz Gráfica de Usuario (GUI)**. En concreto, se propone crear una **aplicación basada en ventanas** que contará con una **interfaz sencilla**. Esta contará con **tres botones principales**: uno para obtener información sobre la aplicación, otro para realizar el reconocimiento de los signos introducidos por el usuario y un tercer botón destinado a la traducción de textos en español al lenguaje de signos correspondiente.

Esta aplicación permitirá a los usuarios interactuar de manera sencilla y efectiva, facilitando la entrada de texto o la realización de signos en el lenguaje de signos español para, posteriormente, obtener su traducción en el formato deseado, ya sea texto a signos, signos a texto o texto a voz.

El desarrollo de la aplicación requiere la **importación** de una serie de **módulos clave**, entre los que se incluyen OpenCV, Numpy, TKinter, Pyttsx3, TensorFlow y Keras. Además, se definen algunas **variables globales**, como las dimensiones de la ventana de la aplicación y el identificador de la cámara.

En este sentido, la clase **'Tk_Manage'** se configura como una **subclase de 'tk.Tk'**, la clase raíz para una **aplicación Tkinter** [52]. Esta clase ha sido diseñada con el objetivo de gestionar el cambio entre las diferentes secciones o páginas de la aplicación, las cuales incluirán una página de inicio, una de información, una para el reconocimiento y otra para la traducción.

La **estructura de la clase 'Tk_Manage'** se basa en **dos métodos:** el inicializador y otro para cambiar la página que se muestra en la aplicación. El **método inicializador** se encarga de iniciar la clase padre Tkinter, crear un nuevo marco que actuará como contenedor de los demás marcos de la aplicación, configurar la cuadrícula donde se ubicarán los marcos y permitir que los elementos de la cuadrícula se expandan. Además, crea un diccionario vacío para almacenar las referencias a todos los marcos y configura la página de inicio como la que se mostrará por defecto.

El **segundo método** facilita el cambio de página mostrada en la aplicación. Para ello, recibe una clase de marco como argumento y utiliza **'tkraise()'** para llevar al frente el marco correspondiente.

En resumen, la **clase 'Tk_Manage'** es el motor de la aplicación, permitiendo administrar y alternar entre los diferentes marcos o secciones que la componen.

La **página de inicio** se trata de un **subtipo de 'tk.Frame'**, un widget de la biblioteca de interfaz gráfica de usuario Tkinter. En esencia, esta clase está diseñada para representar la página de inicio en la aplicación del Lenguaje de Signos Español.

El **constructor inicial de la clase de la página de Inicio** acepta **dos parámetros**: el **widget contenedor de la página de Inicio** y **'controller**', que se trata de una instancia del motor de la aplicación. Esta última, también se encarga de la transición entre diferentes páginas de la aplicación.

Posteriormente, se establecen **tres botones**: uno para la página de **"Reconocimiento",** otro para la página de **"Traducción"** y un tercer botón para la página de **información de "?"**. Se emplea

el **método 'show_frame()'** del controlador para cambiar entre las páginas cuando se presiona el botón.

La clase de la **página de Inicio** proporciona la funcionalidad de navegar entre las diferentes páginas de la aplicación. A través de la **Figura *20,*** se puede ver el resultado de la página de Inicio.

Figura 20. *Interfaz de la página de Inicio*

Si se presiona el **botón "?"**, navega hasta una página en la que proporciona detalles acerca de la funcionalidad de la propia aplicación y permite al usuario volver a la página de inicio.

Figura 21. *Interfaz de la página de Información*

En caso de que se presione el botón de **"Traducción"**, se navegará a su correspondiente página, la cual es heredada de tk.Frame, un componente de la biblioteca de interfaz de usuario. Para desarrollar la funcionalidad de la ventana, se establece el método que inicializa a la clase y establece varias propiedades. Se crea un campo de entrada para que los usuarios puedan introducir texto. Este campo está configurado para responder a ciertos eventos. Además, se crea un botón que redirige al usuario a la página de Inicio cuando se pulsa.

El desarrollo para crear un **sistema de traducción de texto a signos** es cuando el usuario presiona la tecla de **retorno** en el campo de entrada, se activa el **método get_input**. Este método borra cualquier etiqueta de imagen existente, obtiene el texto del campo de entrada y luego llama a los **métodos word2faces y procesadoImagenes** para convertir el texto en una serie de imágenes que representan letras en el Lenguaje de Signos Español.

El **método word2faces** toma una cadena de texto, la convierte en minúsculas, elimina los signos de puntuación y los espacios, y devuelve una lista de strings. Cada string corresponde a una letra en el texto original y tiene un formato "lse-" seguido de la letra que le corresponde.

El **método procesadoImagenes** toma la ruta de la imagen, la abre, la redimensiona, la convierte en una imagen de Tkinter, la coloca en una etiqueta de imagen y luego devuelve esa etiqueta.

En general, el **sistema de traducción** funciona de la siguiente manera: los usuarios ingresan el texto y luego observan cómo ese texto se representa en la dactilología del Lenguaje de Signos Español a través de una serie de imágenes. A través de la **Figura *22***, se puede ver el **diseño de la interfaz** y en la **Figura *23***, se puede ver el **resultado**.

Figura 22. *Interfaz de la página de Traducción*

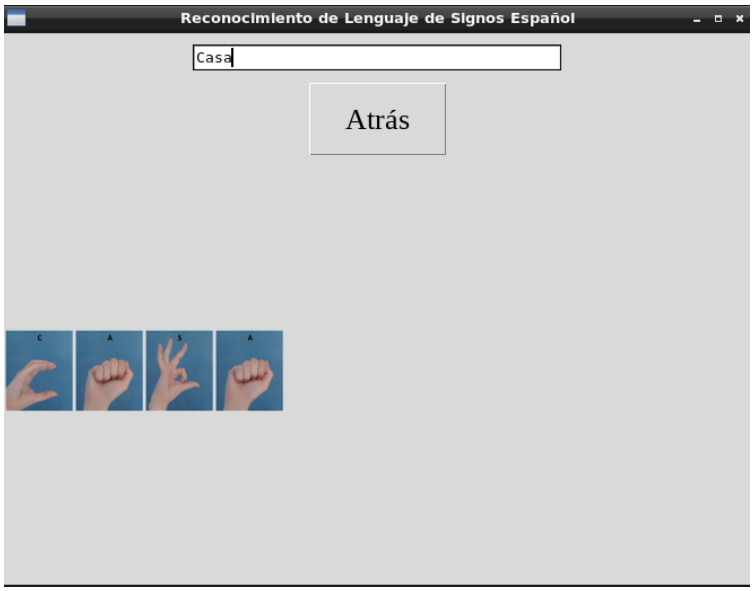

Figura 23. *Interfaz de la página de Traducción a partir de la palabra "casa"*

En caso de que presione el botón de **"Reconocimiento"**, se navegará hasta la clase en la que se ejecuta la funcionalidad de reconocimiento e inicializa la misma.

Durante la **inicialización de la clase**, se configuran varios elementos de la interfaz de usuario, conocidos como **widgets**. En este caso, hay **cuatro widgets**: tres botones y un cuadro de texto. El primer botón, **"Inicio Webcam"**, está configurado de manera que al hacer clic en él se llama a la **función show_camera**, es la encargada de abrir la webcam y capturar el video. Para ello, utiliza la biblioteca cv2 (OpenCV) para abrir la cámara y mostrar el video en una ventana, mientras que keras, una biblioteca de aprendizaje profundo se utiliza para cargar el modelo previamente entrenado para reconocer la dactilología del Lenguaje de Signos Español.

En la función, se define un área de interés (ROI) en el video capturado. Este ROI se dimensiona a un tamaño de 56x56 y se normaliza dividiendo por 255, que es el valor máximo de un píxel en una imagen digital, lo que trae los valores de los píxeles a un rango entre 0 y 1. Luego, el modelo hace una predicción del signo que se está realizando en la ROI. La letra correspondiente a este signo se superpone en el cuadro de video.

Por otro lado, el segundo elemento widget de la interfaz es un **cuadro de texto,** que se utiliza para mostrar texto, la ROI se captura cada 10 segundos. Si la letra que el modelo predijo es diferente a la que se registró previamente, se inserta en el cuadro de texto. Este ciclo de captura de video continúa hasta que se presiona 'q', momento en el que se libera la captura y se cierra la ventana de video.

Además, se agrega otro botón **"Convertir a Voz"**, que al ser presionado llama a la **función text_to_speech**, esta función se encarga de convertir el texto que se introdujo a partir del reconocimiento de texto a voz.

Finalmente, se agrega un **botón** que al ser presionado **lleva al usuario de vuelta a la página de inicio.**

En conclusión, esta clase está diseñada para que los usuarios puedan capturar desde la webcam a tiempo real un video de sus signos a partir de la dactilología del Lenguaje de Signos Español, utilizar un modelo previamente entrenado para interpretar estos gestos como letras, mostrar estas letras en un cuadro de texto y posteriormente convertirlas en voz de manera opcional. A través de la **Figura *24***, se puede ver el **diseño de la interfaz de usuario.**

Figura 24. *Interfaz de la página Reconocimiento*

7. Pruebas

En este apartado, se presentan las pruebas que se han realizado para verificar que el entrenamiento y la solución basada en aprendizaje profundo para el reconocimiento del lenguaje de signos español funciona correctamente y cumple con los requisitos previamente requeridos para su desarrollo.

7.1. Pruebas de Funcionalidad

Se debe garantizar que el modelo, tras haber sido entrenado utilizando el conjunto de datos de entrenamiento, pueda predecir de manera precisa los datos de test. Para ello, se han implementado diferentes métricas evaluativas que permiten analizar exhaustivamente el rendimiento del modelo. Estas métricas incluyen exactitud (Accuracy), función de pérdida (Loss), precisión, sensibilidad (Recall) y la puntuación F1 [53] Scikit Learn. (Todas estas métricas se calcularon durante el entrenamiento del modelo para informar sobre el progreso del entrenamiento.

7.1.1. Accuracy

La **exactitud** (accuracy) es una de las **métricas** más utilizadas para **evaluar el rendimiento de un modelo de clasificación de imágenes**. Mide la proporción de predicciones correctas hechas por el modelo en relación con todas las predicciones realizadas. En la **Fórmula *1***, se puede ver la **fórmula para calcular la exactitud** (accuracy).

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)}$$

Fórmula 1. *Accuracy*

Las **clases predichas correctamente por el modelo** se conocen como **Verdaderos Positivos (TP)** para cada una de las clases en particular. Es decir, cuando el modelo acierta al predecir una instancia de una determinada clase, se considera un TP para esta clase.

Por otro lado, cuando el **modelo predice una instancia como perteneciente** a una clase especifica, **pero en realidad pertenece a otra**, se denomina **Falso Positivo (FP)** para la clase que fue erróneamente identificada. Sin embargo, se puede considerar como **Falso Negativo (FN) para la clase real**, ya que se perdió la oportunidad de clasificar correctamente esta instancia.

Además, para cada clase, los **Verdaderos Negativos (TN)** representan las instancias que pertenecen a las demás clases y **fueron correctamente clasificadas como no pertenecientes** a la clase de interés.

Así, en un escenario de clasificación multiclase, cada una de las clases tiene sus propios TP, FP, TN y FN, lo que permite una evaluación más detallada del desempeño del modelo.

Durante el **proceso de entrenamiento**, se ha optado por emplear **accuracy** para conservar el mejor rendimiento. Si la exactitud alcanzada por el modelo en la época actual supera las cifras registradas en todas las épocas anteriores, entonces el modelo se guarda como el "mejor" modelo.

A través del entrenamiento del modelo, se puede observar en la **Figura *25*** la evolución del entrenamiento. Si se analiza la **gráfica**, se puede ver que, para los datos de entrenamiento, el accuracy del modelo mejora drásticamente desde la primera época (3,5 %) hasta la décima (98,35 %), y luego sigue mejorando gradualmente hasta llegar a 99,38 % de acierto. Esto sugiere que el modelo está aprendiendo bien los datos de entrenamiento. El hecho de que el accuracy en el conjunto de test también siga el mismo patrón que en el conjunto de entrenamiento es muy positivo, ya que indica que el modelo no está simplemente memorizando los datos de entrenamiento (lo que se conoce como "overfitting"), sino que también es capaz de generalizar a datos nuevos.

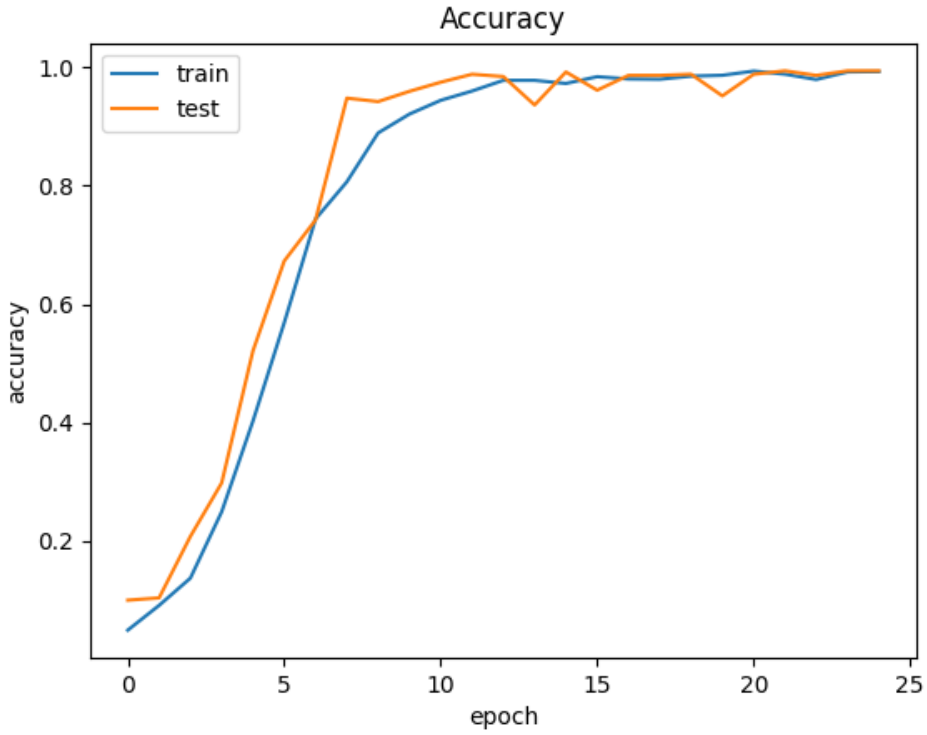

Figura 25. *Gráfico de Accuracy*

En general, estos **resultados son bastantes altos**, especialmente si se considera que inicialmente se partía de un rendimiento bastante modesto. Se puede decir que, para abordar el problema en estas circunstancias particulares, estos niveles de rendimiento son más que suficientes.

Sin embargo, es crucial comprender la **función de la pérdida del modelo neuronal**. Esta función actúa como un indicador de qué tan acertadas son las predicciones de este modelo en comparación con los resultados reales. Hablando técnicamente, la función de pérdida cuantifica la discrepancia entre las estimaciones del modelo y los valores reales. La minimización de esta discrepancia es precisamente el propósito del proceso de entrenamiento del modelo.

Durante el entrenamiento, se enfoca en ajustar los pesos y sesgos de la red neuronal con el **objetivo** de **reducir el valor de la función de pérdida**. Esta optimización se logra generalmente a través de un algoritmo de optimización. En el caso de este Trabajo Final de Grado, se ha utilizado el **optimizador de descenso de gradiente estocástico (SGD)**.

7.1.2. Función de Pérdida

La **función de pérdida** utilizada para este algoritmo es la **'categorical_crossentropy'**, comúnmente utilizada en problemas de clasificación de múltiples clases, la dactilología del Lenguaje de Signos Español está compuesto por 26 letras (26 clases). Esta es una forma de función de pérdida de entropía cruzada, que mide la disimilitud entre la distribución de probabilidad predicha por el modelo y la distribución de probabilidad real.

En términos más sencillos, la función de pérdida utilizada en esta solución penaliza al modelo si predice una probabilidad baja para la clase verdadera. Cuanto más baja es la probabilidad que el modelo asigna a la clase verdadera, mayor es el valor de la pérdida. A través de la **Fórmula *2***, se puede ver la **fórmula de la función de pérdida categorical cross entropy**.

$$CE = -\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{c} y_i \log \widehat{y}_i$$

Fórmula 2. *Función de Pérdida Cross Entropy*

Para una mejor comprensión de los **términos utilizados en la evaluación de un modelo neuronal**, es crucial definirlos:

El **término 'i'** se refiere a **la clase**. En este contexto, una clase es una letra del conjunto dactilológico que puede predecir el modelo. Por ejemplo, si se está clasificando un signo, cada tipo de signo sería una clase.

El **símbolo 'c'** denota **el número total de clases en el problema**. En este caso, el conjunto dactilológico del Lenguaje de Signos Español está compuesto por 26 letras, entonces 'c' sería igual a 26.

'Y_i' se refiere a **la clase real para la i-ésima instancia de los datos.** Este es el valor que se conoce como correcto, obtenido de los datos de entrenamiento.

Finalmente, **'$\widehat{y}_i$' representa la clase predicha por el modelo para la i-ésima instancia.** Esta es la salida del modelo basado en su aprendizaje y representa la clasificación del signo de la instancia actual.

Al **comparar 'y_i' y '$\widehat{y}_i$'**, se puede evaluar qué tan bien está funcionando el modelo, y es precisamente aquí donde la función de pérdida juega un papel fundamental.

Tras finalizar el entrenamiento del modelo neuronal a partir del conjunto de imágenes, el **resultado** se puede ver en la **Figura *26***. Interpretando el **gráfico de desarrollo** se puede observar

una disminución constante de la pérdida en los datos de entrenamiento desde 4,510 en la primera época hasta la época 25 con un 0,8587. Esto indica que el modelo está aprendiendo y mejorando su rendimiento a lo largo de las épocas en los datos de entrenamiento.

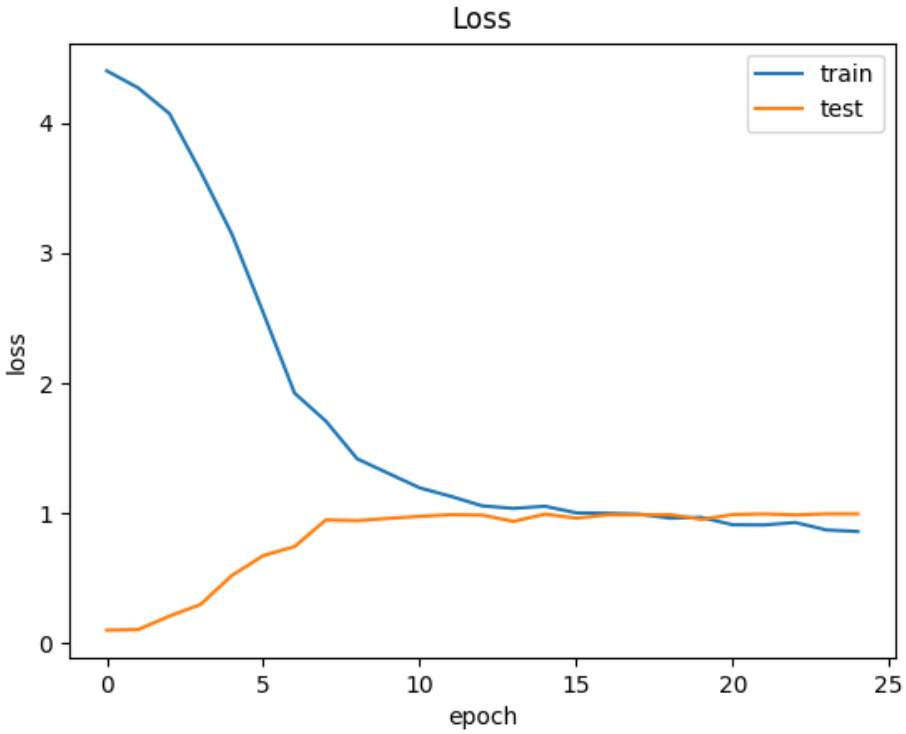

Figura 26. *Gráfico de Función de Pérdida Cross Entropy*

Por otro lado, para los **datos de test**, la pérdida comienza en 0,100 en la primera época, que es bastante baja, lo que es buena señal. Sin embargo, la pérdida aumenta hasta 0,7887 en la séptima época y luego crece lentamente hasta 0,8494 en la época 25. Este patrón sugiere que el **modelo podría estar sobreajustando ('overfitting') los datos de entrenamiento**.

Es normal que haya cierta discrepancia entre la pérdida de entrenamiento y la de test, ya que el modelo no ha visto los datos de test durante el entrenamiento. Sin embargo, un aumento de la pérdida de test durante el entrenamiento es un indicador de sobreajuste, aunque en este caso no es muy severo.

Tener cierto grado de sobreajuste es casi inevitable en el aprendizaje profundo, por lo que aparentemente no es algo preocupante para la funcionalidad del algoritmo.

Existen otras **métricas** a considerar para **verificar la funcionalidad del algoritmo**, tales como la **exhaustividad (recall), la precisión y la puntuación F1**. Estas métricas proporcionarán una imagen más completa del rendimiento del modelo.

7.1.3. Recall

La **exhaustividad (recall),** es una **métrica** utilizada para evaluar la capacidad de un modelo para identificar todos los casos relevantes en un conjunto de dato**s**. Es particularmente útil en situaciones donde los falsos negativos son más problemáticos que los falsos positivos.

En el caso de un algoritmo de aprendizaje profundo para el reconocimiento dactilológico del Lenguaje de Signos Español, la exhaustividad **mediría la capacidad del algoritmo para identificar correctamente todos los signos que aparecen en un conjunto de datos**.

En la **Fórmula *3***, se puede ver la **fórmula para calcular la exhaustividad**.

$$Recall = \frac{TP}{TP + FN}$$

Fórmula 3. *Recall*

Los **Verdaderos Positivos (TP)** son los signos que el algoritmo identificó correctamente. En cambio, **los Falsos Negativos (FN)** son los signos que el algoritmo no logró identificar.

Lo ideal, es tener una **alta exhaustividad (recall)**, lo que significa que es bueno para identificar todos los signos relevantes en el conjunto de datos.

En la **Figura *27***, se puede observar la **gráfica de exhaustividad del algoritmo**.

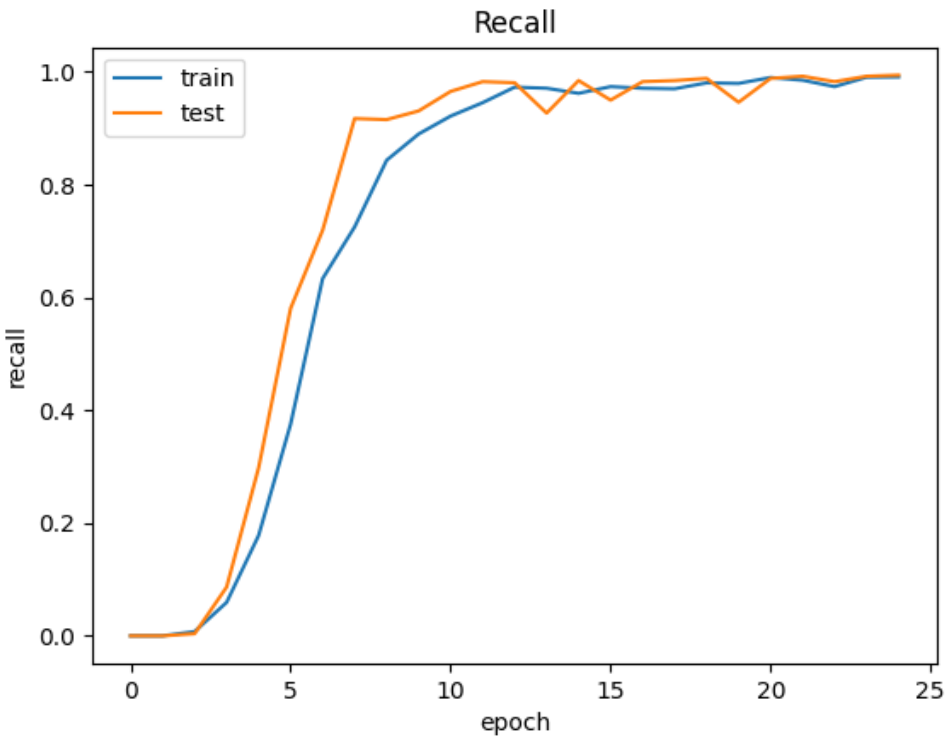

Figura 27. *Gráfico de Recall*

En este caso, en los datos de entrenamiento se muestra que el modelo mejora su rendimiento con cada época, empezando con un recall de 0 en la época 0 y alcanzando un recall de 0,9910 en la época 25. Esto indica que el modelo está aprendiendo efectivamente a identificar los casos positivos en los datos de entrenamiento.

En cuanto a los datos de test, el modelo también mejora su rendimiento con el tiempo, pero con algunos altibajos. El recall aumenta desde 0 en la época 0 hasta 0.975 en la época 5, y después hay algunas fluctuaciones hasta que se estabiliza en la época 20 y alcanza un recall de 0.9920 en la época 25.

Estas fluctuaciones podrían deberse a la naturaleza estocástica del aprendizaje profundo del modelo neuronal. En cualquier caso, el hecho de que el recall final en los datos de test sea alto y muy similar al de los datos de entrenamiento es una buena señal, ya que indica que el **modelo es capaz de generalizar bien los nuevos datos**. Sin embargo, sería útil examinar la precisión y el F1 score para tener una imagen más completa del rendimiento del modelo.

7.1.4. Precisión

La **precisión** es una **métrica** que mide el rendimiento de un modelo de clasificación. En el contexto del reconocimiento dactilológico del Lenguaje de Signos Español, que implica reconocer signos para interpretar el lenguaje de signos, la precisión podría usarse para **evaluar la proporción de signos que el modelo identifica correctamente de todos los que clasifica**.

En términos formales, la precisión se calcula como el número de verdaderos positivos (VP), el número de gestos que el modelo clasifica correctamente, dividido por la suma de los verdaderos positivos y los falsos positivos (FP), el número de signos que el modelo clasifica incorrectamente. A través de la **Fórmula *4***, se puede ver la **fórmula de la precisión.**

$$precision = \frac{VP}{VP + FP}$$

Fórmula 4. *Precisión*

Un modelo con una precisión alta en el reconocimiento de signos del Lenguaje Español sería uno que, cuando predice un signo, es muy probable que esté en lo cierto.

Dicho esto, se **analizará el gráfico de rendimiento** a través de la **Figura 28**. En los datos de entrenamiento, el modelo comienza con una precisión de 0 en la época 0 y aumenta gradualmente hasta alcanzar una precisión de 0.9943 en la época 25. Este aumento constante indica que el modelo está mejorando su capacidad para hacer predicciones correctas a lo largo del tiempo en los datos de entrenamiento.

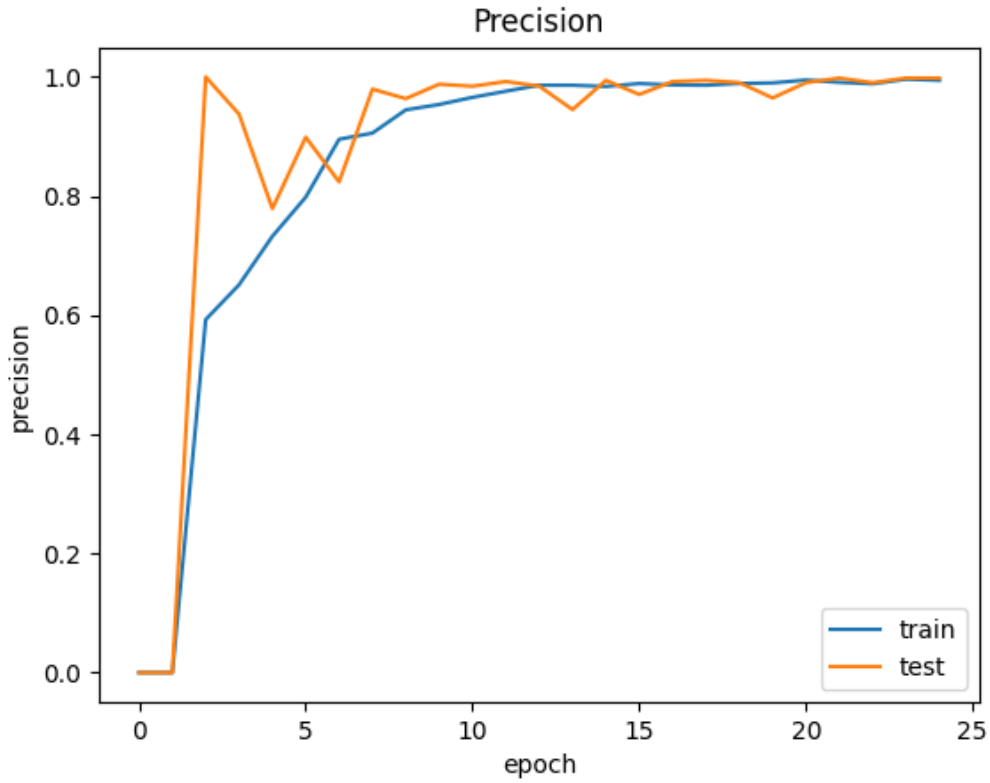

Figura 28. *Gráfico de Precisión*

Por otro lado, en los datos de test, el modelo tiene un comportamiento más fluctuante. Aunque comienza con una precisión muy alta de 0.9998 en la época 1, la precisión disminuye y luego fluctúa con subidas y bajadas antes de estabilizarse en una precisión de 0.9981 en la época 25.

7.1.5. F1 Score

La **puntuación F1,** es una **métrica** que combina la precisión y la exhaustividad (recall) para proporcionar una medida única de la calidad de un modelo de clasificación. A través de la **Fórmula *5***, se puede observar la **fórmula para calcular la puntuación F1**.

$$F1\ Score = \frac{2\ x\ (Precision\ x\ Recall)}{Precision + Recall}$$

Fórmula 5. *F1 Score*

En este caso, al estar describiendo un problema de reconocimiento dactilológico del Lenguaje de Signos Español utilizando aprendizaje profundo, se trata de un problema de clasificación multiclase en el que cada clase representa un signo diferente.

La puntuación F1 puede ser útil para **medir cómo de bien es capaz el modelo de reconocer cada signo**. Por ejemplo, si el modelo tiene una alta precisión, pero una baja exhaustividad para un

signo específico, esto podría significar que el modelo es muy bueno para reconocer ese signo cuando lo ve, pero se pierde muchas instancias de ese signo. En contraste, una alta exhaustividad, pero una baja precisión podría indicar que el modelo identifica muchas instancias del signo correctamente, pero también clasifica incorrectamente muchas instancias que no son de ese signo.

La **Figura *29*** ilustra la **puntuación F1 obtenida durante el entrenamiento del algoritmo**. Tras un meticuloso análisis de las métricas evaluativas de recall y precisión, se anticipa que la **puntuación F1 para el entrenamiento del modelo neuronal será positivamente satisfactoria**.

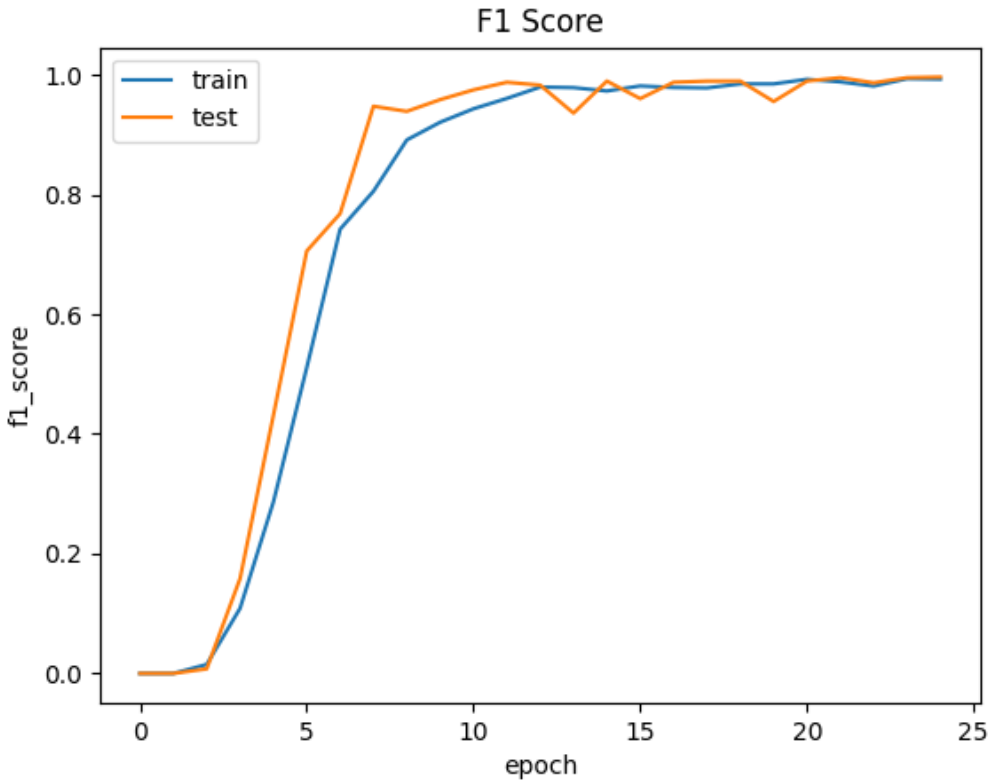

Figura 29. *Gráfico F1 Score*

Aun así, **analizando** el **gráfico**, se puede observar que los **resultados son bastante buenos**. Ambas puntuaciones F1, tanto para los datos de entrenamiento como para los datos de test, son altas, lo que sugiere que el modelo tiene un buen rendimiento general. Aunque hay una variabilidad en las puntuaciones de test entre las épocas 13 y 20, finalmente se estabiliza y supera la puntuación de entrenamiento en la época 25, lo cual es positivo.

Además, es bueno señalar que la puntuación F1 para los datos de test iguala a la de los datos de entrenamiento en la época final, lo que podría indicar que el modelo está generalizando bien y no sufre de sobreajuste (overfitting) severo.

Tras haber analizado todas las métricas, mostrando buenos resultados tanto en el conjunto de entrenamiento como en el de test, se observan indicios bastante sólidos de que el **modelo está funcionando bien y generalizando adecuadamente a datos no vistos**.

7.2. Pruebas de Validación con el usuario

En este tipo de evaluaciones, la **solución basada en aprendizaje profundo para la dactilología del Lenguaje de Signos Español** se someterá a diversas circunstancias para **determinar si satisface las expectativas**. Esto podría plantear desafíos para el modelo en términos de predecir imágenes en tiempo real de manera eficiente.

La primera prueba de validación intentará demostrar que, **ante la interacción de objetos con el entorno, el algoritmo deberá ser capaz de predecir con precisión el signo proporcionado por el usuario**. En este contexto, se ha empleado un cuadro específico ubicado en la región de interés a través del cual el usuario debería poder traducir correctamente el signo a la letra correspondiente.

Después de ejecutar el algoritmo [54] Rubén Tercero [rubentercero869] (16 de julio de 2023). *GrabacionCuadro*. [Vídeo]. Youtube. , se constata que su funcionamiento es adecuado. Para añadir una capa de complejidad, se **evaluará el rendimiento del modelo neuronal en condiciones de iluminación variables para testear su robustez**. El algoritmo deberá demostrar su capacidad para predecir con exactitud lo que el usuario intenta comunicar a través de la dactilología del Lenguaje de Signos Español.

Se puede apreciar que el algoritmo cumple satisfactoriamente con la generalización [55] . A continuación, se realizará una **prueba final con un usuario real para verificar la funcionalidad práctica del algoritmo**.

Una vez realizada la prueba final [56] Rubén Tercero [rubentercero869] (16 de julio de 2023). *GrabacionArantxa*. [Vídeo]. Youtube. , se observa que el algoritmo es capaz de predecir con precisión las diferentes dificultades en un entorno real para reconocer la dactilología del Lenguaje de Signos Español.

7.3. Pruebas de Carga

Las **Pruebas de Carga** están diseñadas para evaluar el comportamiento del sistema cuando se encuentra sometido a cargas elevadas, permitiendo determinar su capacidad de respuesta y su estabilidad.

Para llevar a cabo la prueba de carga del modelo, se mide el tiempo que el sistema emplea en ejecutar determinadas operaciones. En este caso, se enfocará específicamente en la predicción del modelo. El **objetivo** es **minimizar cualquier retraso que pueda ocurrir durante la clasificación realizada por el modelo neuronal**, el cual procesa imágenes en tiempo real capturadas directamente desde el usuario.

Para ello, se utilizará el **módulo 'time' de Python** [57] Python. (25 de junio de 2023). *Time Access and conversions*. , que proporciona varias funciones relacionadas con el tiempo. A través de la **Tabla 3**, se pueden ver las **últimas 10 operaciones para analizar el signo realizado por el usuario junto con la obtención de la duración de cada predicción**.

Número	Predicción
1	0,311 segundos
2	0,342 segundos
3	0,3233 segundos
4	0,3186 segundos
5	0,3106 segundos
6	0,3201 segundos
7	0,3114 segundos
8	0,319 segundos
9	0,3319 segundos
10	0,3104 segundos

Tabla 3. *Duración de cada predicción*

A partir de la **tabla,** se puede interpretar que la solución basada en aprendizaje profundo para el reconocimiento dactilológico del Lenguaje de Signos Español es **capaz de realizar una predicción de la letra que corresponde a un signo en un tiempo muy corto (entre 0,31 y 0,33 segundos).** Este tiempo de respuesta tan rápido es esencial para un sistema de reconocimiento en tiempo real, ya que permite que el proceso de traducción sea fluido y no cause demoras innecesarias para el usuario.

En el **código del algoritmo**, existe la condición de que **la letra solo se traduzca si el modelo hace la misma predicción en 10 frames consecutivos**, es una medida de control de calidad para garantizar la precisión de la traducción. Esto significa que, para que la letra se traduzca, el modelo de aprendizaje profundo tiene que estar bastante seguro de su predicción, ya que tiene que hacer la misma predicción en varios frames consecutivos.

Si bien esta medida puede ayudar a evitar errores de traducción, hace que aumente el tiempo necesario para que se traduzca una letra. Por ejemplo, si el modelo toma una media de 0,32

segundos para hacer una predicción, entonces necesitará alrededor de 3,2 segundos (0,32 segundos/frame x 10 frames) para traducir una letra.

El balance entre precisión y rapidez es un desafío común en el desarrollo de aplicaciones en tiempo real.

7.4. Pruebas de Eficiencia

Las **Pruebas de Eficiencia** buscan **cuantificar el consumo de recursos del sistema, como memoria, CPU y GPU, durante la ejecución normal**. Esta medida de eficiencia es **importante para garantizar que el sistema pueda funcionar de manera óptima**, incluso en los dispositivos con recursos limitados.

Para **realizar las pruebas de eficiencia del sistema**, se necesitaría **ejecutar y monitorizar el uso de la CPU y de la memoria.** Para ello, se utilizará la **biblioteca 'psutil' de Python** [58], que proporciona una **API** para recuperar información sobre el uso de recursos del sistema. En cambio, para **monitorizar la GPU**, se ha utilizado la **herramienta 'tegrastats'**.

Para monitorear los recursos se crea un hilo separado. Esto significa que este hilo correrá en paralelo con el programa principal. Los **resultados aproximados del uso de la CPU han sido de un 57-59%, de la memoria 76-78% y de la GPU un 50-60%**.

Los resultados apuntan a que el **algoritmo se desempeña eficientemente**. El consumo moderado de la CPU indica que, aunque el algoritmo requiere algunos recursos para su procesamiento, no ejerce una presión excesiva sobre la misma. El uso elevado de la memoria podría implicar que el modelo de aprendizaje profundo gestiona un volumen considerable de datos de entrada. Por otro lado, la carga de trabajo de la GPU es notablemente baja.

En términos generales, parece que el algoritmo de reconocimiento del Lenguaje de Signos se está ejecutando de manera eficiente en la NVIDIA Jetson Nano, esto es un resultado bastante optimista, ya que se trata de un dispositivo de baja potencia en comparación con muchos otros PCs. A continuación, se **ejecutará el modelo con múltiples aplicaciones ejecutándose simultáneamente para ver la eficiencia del modelo**.

En esta oportunidad, el modelo ha sido puesto a prueba en un entorno con múltiples pestañas en funcionamiento, algunas de ellas reproduciendo vídeos a través del navegador web, generando así una considerable demanda de recursos en segundo plano.

Los resultados de la ejecución del algoritmo han sido negativos, el uso de la CPU se situaba en un rango de 19-20%. Este uso es bastante bajo, lo que sugiere que el **algoritmo en sí no está poniendo una carga significativa en la CPU**. Sin embargo, el hecho de que el rendimiento haya sido bajo puede indicar que la CPU está siendo utilizada intensamente por otras tareas en segundo plano.

El **uso de la memoria (81-87%)** sugiere que hay una **gran demanda de recursos de memoria**, que podrían estar siendo consumidos por el algoritmo, pero también por las múltiples pestañas

abiertas. Es posible que la falta de memoria disponible esté afectando negativamente el rendimiento del algoritmo.

El **uso de la GPU crece hasta el 60%,** es probable que la reproducción de videos reproducidos en segundo plano pudiese estar limitando los recursos del algoritmo.

Por lo tanto, para que el **algoritmo sea ejecutado correctamente** es necesario la **utilización de un hardware más potente** que pudiera manejar estas múltiples demandas o cerrar las pestañas durante la ejecución del algoritmo.

8. Conclusiones

En el desarrollo de este Proyecto Final de Grado, se ha propuesto como objetivo primordial la **creación y puesta en marcha de un algoritmo de reconocimiento y traducción de la dactilología del Lenguaje de Signos Español en tiempo real**, fundamentado en **técnicas de aprendizaje profundo**. Este algoritmo se distingue por su **precisión y fiabilidad** en la interpretación de las letras del alfabeto, permitiendo así la formación de palabras completas a partir de estas letras, las cuales pueden ser traducidas a lenguaje oral o escrito.

Para alcanzar con éxito este ambicioso objetivo, se han aplicado **técnicas de aumentación de datos y aprendizaje por transferencia**, dando como resultado un modelo neuronal eficaz y ajustado a los requisitos previamente establecidos.

Después de la etapa de diseño, codificación y pruebas, se puede afirmar que se ha logrado cumplir con el objetivo principal. El algoritmo creado ha probado su eficiencia en la **identificación y traducción precisa de las letras de la dactilología del Lenguaje de Signos Español**. Además, ha demostrado su habilidad para **formar palabras a partir de las letras identificadas** y su correcta **interpretación al lenguaje oral o escrito**.

En adición, se ha logrado el **objetivo secundario** de este proyecto que consistía en la **creación de un traductor funcional como herramienta de asistencia** para aquellos interesados en aprender la dactilología del Lenguaje de Signos Español o que necesiten comunicarse con personas con discapacidades auditivas o del habla. Para conseguir esto, hemos desarrollado un **método en Python**, posicionándonos así un paso más cerca de una sociedad más inclusiva.

A lo largo del proyecto, se logró desarrollar un algoritmo alternativo que era capaz de reconocer mi mano al hacer gestos dactilológicos del Lenguaje de Signos Español. Sin embargo, al realizar pruebas con otros usuarios, la metodología inicialmente empleada para desarrollar dicho algoritmo no dio los resultados esperados. A pesar de que las pruebas realizadas en mi mano fueron exitosas, el algoritmo no lograba una generalización adecuada debido a la diversidad en las características de las manos de diferentes individuos.

El enfoque alternativo en extraer los puntos clave resultó insuficiente para adaptarse a este espectro de variabilidad, lo que implicó que el algoritmo no se comportara de manera óptima en manos distintas a la mía. Este obstáculo nos llevó a replantear y adaptar nuestra estrategia a lo largo del proyecto, en búsqueda de una solución más generalizable.

Por lo tanto, se introdujo un nuevo objetivo para este proyecto: ajustar y perfeccionar el algoritmo para que fuese capaz de funcionar eficientemente en un rango más amplio, en lugar de limitarse a una única instancia. Este reto significó un nuevo cambio de enfoque y metodológico, pero fue crucial para mejorar la aplicabilidad y utilidad del algoritmo.

En términos de competencias demostradas, la elaboración de una solución para el reconocimiento del Lenguaje de Signos Español basado en el aprendizaje profundo, demuestran una sólida comprensión de la **programación en Python**, la **manipulación de imágenes** utilizando **OpenCV y Pillow**, la **creación y entrenamiento** del modelo de aprendizaje profundo utilizando **TensorFlow** y la **construcción de interfaces de usuario** con **Tkinter**. Además, a lo largo del desarrollo de este, se muestra cómo **estas tecnologías se pueden integrar en una única aplicación** para el reconocimiento dactilológico del Lenguaje de Signos Español.

A lo largo del desarrollo de este Trabajo Final de Grado, he tenido la oportunidad de **adquirir y profundizar conocimientos** en diversos campos que desconocía o sobre los que tenía una base muy limitada.

Primero, he aprendido mucho sobre la **comunidad sorda** y la **cultura del Lenguaje de Signos Español**, lo que me ha permitido comprender mejor las dificultades que enfrentan estas personas en su día a día, y cómo la tecnología puede contribuir a superar algunas de estas barreras. Además, este entendimiento ha sido esencial para definir los requerimientos y funcionalidades de la solución propuesta, a fin de que realmente responda a las necesidades de los usuarios potenciales.

En cuanto a las **tecnologías**, a lo largo del desarrollo del trabajo, he logrado mejorar mi habilidad para la utilización de diversas **librerías de Python** como **TensorFlow y Keras** para la **creación, entrenamiento y validación de redes neuronales profundas**.

También, me he familiarizado con el **preprocesamiento de imágenes**. Esto incluye la normalización de las imágenes, la ampliación del conjunto de datos a través de técnicas de aumentado y la división del conjunto de entrenamiento, validación y prueba.

Finalmente, el proyecto también me ha permitido desarrollar mis habilidades en la **gestión de proyectos**. Planificar, coordinar y seguir el progreso de un proyecto de esta magnitud a lo largo de varios meses, ha sido un reto en sí mismo, pero he aprendido mucho sobre cómo manejar eficazmente el tiempo y los recursos disponibles.

En el **proceso** de desarrollar una solución basada en el aprendizaje profundo para el reconocimiento de la dactilología del lenguaje de signos español, se han experimentado una variedad de **desafíos**.

Uno de los mayores retos de este proyecto fue la **recopilación de un conjunto de datos lo suficientemente grande y diverso para entrenar el modelo de aprendizaje profundo**. No fue posible encontrar fuentes accesibles y de alta calidad de imágenes del Lenguaje de Signos Español, por lo que se optó por desarrollar un conjunto de imágenes propias [59]. Algo que resultó útil pero no suficiente.

Para resolver la problemática de los datos, se tuvo que recurrir a **técnicas de aumento**. Esto ayudó a generalizar el modelo, la falta de una amplia gama de datos en el Lenguaje de Signos Español es una limitación que debe reconocerse.

Otro problema significativo, fue el **diseño del modelo y el ajuste de sus hiper parámetros para optimizar el rendimiento**. A través de varias pruebas y de diferentes enfoques, se logró diseñar un modelo neuronal con un gran rendimiento.

Cabe señalar que, a lo largo del desarrollo, se han cometido muchos **errores**. Un error destacable fue subestimar la cantidad de **tiempo** que tomaría el utilizar distintos diseños de redes neuronales y lo que implicaría en el preprocesamiento de los datos para cada modelo utilizado. Si hubiera planeado mejor desde el principio la importancia de este proceso habría sido menos costoso.

En general, el desarrollo de esta solución ha sido una experiencia gratificante y enriquecedora que me ha permitido aplicar y profundizar mis habilidades técnicas, a la vez que ha aumentado mi apreciación por el Lenguaje de Signos Español y la necesidad de mejorar la accesibilidad y la inclusión para la comunidad sorda.

8.1. Relación del trabajo desarrollado con los estudios cursados

Durante mi formación académica, he recibido una **sólida base de conocimientos** en áreas como la **inteligencia artificial** y la **visión por computador**. Estos fundamentos teóricos me han proporcionado una comprensión profunda de los algoritmos y técnicas subyacentes en el campo de la inteligencia artificial.

En particular, los **conocimientos** adquiridos a través de las **asignaturas de Visión Por Computador y Machine Learning en Entornos Industriales**, me han permitido aplicar técnicas de procesado de datos y de aprendizaje profundo. Estas asignaturas me han familiarizado con modelos de redes neuronales convolucionales, elaboración de modelos a partir de modelos pre entrenados y procesamiento de datos, que son componentes clave para la elaboración de la solución propuesta.

La elaboración de este Trabajo Final de Grado ha requerido la aplicación y puesta en práctica de diversas competencias transversales adquiridas a lo largo de mi trayectoria como estudiante.

El desarrollo de una solución basada en el aprendizaje profundo para el reconocimiento de la dactilología para el Lenguaje de Signos Español implica enfrentarse a desafíos técnicos y tomar decisiones para superar obstáculos. He aplicado mis **habilidades de resolución de problemas** para identificar y abordar los desafíos que surgieron durante el proceso de desarrollo.

Además, se ha hecho uso de mi **capacidad de análisis y evaluación crítica** para examinar diferentes soluciones propuestas en la literatura científica y seleccionar las más adecuadas para mi proyecto. Además, se ha realizado una evaluación exhaustiva de los resultados obtenidos para identificar posibles mejoras y limitaciones de mi enfoque.

La elaboración de un proyecto de esta índole requiere una gestión eficiente del tiempo para cumplir con los plazos establecidos. He tenido que **planificar y organizar** el trabajo de manera efectiva, distribuyendo las tareas de forma adecuada para garantizar la finalización del proyecto.

En resumen, este Trabajo Final de Grado demuestra la aplicación de un amplio conjunto de conocimientos y habilidades adquiridos a lo largo de mis años de estudio. Mediante el uso de tecnologías avanzadas en el campo del aprendizaje profundo y la integración de competencias transversales, se ha abordado con éxito un problema real y se ha demostrado mi dominio tecnológico en este ámbito.

9. Bibliografía

[1] Fundación CNSE. (8 de enero de 2001). *Dactilológico*. Banco de imágenes y signos LSE. https://www.fundacioncnse.org/educa/bancolse/dactilologico.php#gsc.tab=0

[2] Sánchez, C. (11 de diciembre de 2019). *Actualizaciones en la 7ma (séptima) edición de las Normas* APA. Normas APA (7ma edición). https://normas-apa.org/wp-content/uploads/Guia-Normas-APA-7ma-edicion.pdf

[3] Google. (2014). *Google Colab*. https://colab.research.google.com/?utm_source=scs-index

[4] Amazon. (2017). *Amazon SageMaker Documentation*. https://docs.aws.amazon.com/sagemaker/index.html

[5] Microsoft. (2010). *Azure Machine Learning documentation*. https://learn.microsoft.com/en-us/azure/machine-learning/?view=azureml-api-2

[6] IBM. (2014). *Watson Studio*. https://cloud.ibm.com/developer/watson/documentation

[7] NVIDIA. (2019). *Kit de desarrollo Jetson Nano.* https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-nano-developer-kit/

[8] NVIDIA Corporation. (17 de diciembre de 2019). *Jetson Nano Developer Kit. User Guide*. https://developer.download.nvidia.com/embedded/L4T/r32-3-1_Release_v1.0/Jetson_Nano_Developer_Kit_User_Guide.pdf

[9] Wu, E. (16 de julio de 2020). NVIDIA Jetson Nano and Jetson Xavier NX Comparison: Specifications, Benchmarking, Container Demos, and Custom Model Inference. *Seeed Studio*. https://www.seeedstudio.com/blog/2020/06/04/nvidia-jetson-nano-and-jetson-xavier-nx-comparison-specifications-benchmarking-container-demos-and-custom-model-inference/

[10] NVIDIA Developer. (1 de mayo de 2022). *Jetson Modules*. NVIDIA. https://developer.nvidia.com/embedded/jetson-modules

[11] NVIDIA Developer. (1 de mayo de 2022). *NVIDIA cuDNN*. NVIDIA. https://developer.nvidia.com/cudnn

[12] Rubiales, A. (22 de abril de 2023). Cómo construir tu propio equipo de Inteligencia Artificial y ahorrar miles de euros. *Medium*. https://rubialesalberto.medium.com/como-construir-tu-propio-equipo-de-inteligencia-artificial-y-ahorrar-miles-de-euros-f040da2b80d7

[13] Intel Corporation. (Junio de 2000). *OpenCV*. https://opencv.org/

[14] Ilango, G. (6 de abril de 2017). Hand Gesture Recognition using Python and OpenCV-Part 1. *Gogulilango*. https://gogulilango.com/software/hand-gesture-recognition-p1

[15] Hussein, A. (24 de abril de 2019). *Manual de gestos*. GitHub. Recuperado el 20 de mayo de 2023. https://github.com/adilsofficial/HandGesture

[16] Heintz, B. (17 de diciembre de 2018). Training a neural Network to Detect Gestures with OpenCV in Python. *Towards Data Science*. https://github.com/athena15/project_kojak

[17] Heintz, B. [scgrinchy]. (18 de diciembre de 2018). *Gesture Recognition Demo with Python & OpenCV* [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=kvyIaGgdwio

[18] TensorFlow. (s.f.). *Modelos de TensorFlow.js*. https://www.tensorflow.org/install?hl=es-419

[19] Bazarevsky, V., & Zhang, F. (19 de agosto de 2019). Seguimiento de manos en tiempo real en el dispositivo con MediaPipe. *Google Research*. https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

[20] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, F.Z., Yong, M.G., Lee, J., Chang, W.T., Hua, W., Georg, M., & Grundmann, M. (2019). *MediaPipe: A Framework for Building Perception Pipelines.* https://arxiv.org/pdf/1906.08172.pdf

[21] Bazarevsky, V., & Zhang, F. (24 de diciembre de 2022). *MediaPipe Handpose*. Googleapis. https://storage.googleapis.com/tfjs-models/demos/handtrack/index.html

[22] Leap Motion. (2018). Hand Gesture Recognition Database. *Kaggle*. https://www.kaggle.com/datasets/gti-upm/leapgestrecog

[23] Harrington, B. (20 de julio de 2018). Hand Gesture Recognition Database with CNN. *Kaggle*. https://www.kaggle.com/code/benenharrington/hand-gesture-recognition-database-with-cnn/notebook

[24] Xing, X. (2021). *Pose-detection.* TensorFlow. https://github.com/tensorflow/tfjs-models/tree/master/pose-detection

[25] Xing, X. (24 de diciembre de 2022). *Pose Detection*. Googleapis. https://storage.googleapis.com/tfjs-models/demos/posenet/camera.html

[26] Paszke, A., Soumith, G., & Chanan, G. (2 de febrero de 2017). *PyTorch*. https://pytorch.org/

[27] Chollet, F. (27 de marzo de 2015). *Keras*. https://keras.io/

[28] Tecperson. (2 de agosto de 2017). Sign Language MNIST. *Kaggle*. https://www.kaggle.com/datasets/datamunge/sign-language-mnist/code

[29] Mathur, M. (2020). *CNN usando Keras*. Kaggle. https://www.kaggle.com/code/madz2000/cnn-using-keras-100-accuracy/notebook

[30] Kazi, A. (20 de mayo de 2023). Sign Language Recognition Using OpenCV. *Github.* https://github.com/Arshad221b/Sign-Language-Recognition

[31] Thema, G. (10 de mayo de 2023). Google American Sign Language Fingerspelling Recognition. *Kaggle*. https://www.kaggle.com/competitions/asl-fingerspelling

[32] Signall. (2021). *We translate sign language. Automatically.* https://www.signall.us/

[33] Camgoz, N.C., Saunders, B., Rochette, G., Giovanelli, M., Inches, G., Nachtrab-Ribback, R., & Bowden, R. (2021). *Content4All Open Research Sign Language Translation Datasets*. https://arxiv.org/pdf/2105.02351.pdf

[34] Rodríguez-Moreno, I., Martínez-Otzeta, J.M., Goienetxea, I., & Sierra, B. (24 de noviembre de 2022). Avances en la mejora de la comunicación de las personas con problemas de audición. *Campusa*. https://www.ehu.eus/es/-/problemas-audicion-comunicacion-avances

[35] Villaplana Moreno, A. (2020). *Generación de lengua de signos continua*. [Trabajo Fin de Grado, Universidad Politécnica de Valencia]. RiuNet UPV. https://riunet.upv.es/bitstream/handle/10251/149597/Villaplana%20-%20Generaci%C3%B3n%20de%20lengua%20de%20signos%20continua.pdf

[36] https://www.google.com/search?q=sign+language+recognition&rlz=1C1VDKB_esES1008ES1008&oq=sign+&gs_lcrp=EgZjaHJvbWUqBggAEEUYOzIGCAAQRRg7MgYIARBFGDkyBggCEEUYOzIGCAMQRRg7MgYIBBBFGDwyBggFEEUYPDIGCAYQRRg8MgYIBxBFGDzSAQgxNjc0ajBqN6gCALACAA&sourceid=chrome&ie=UTF-8

[37] Opensource. (6 de junio de 2023). *New developments at Opensource.com* https://opensource.com/

[38] https://www.amazon.es/NVIDIA-Jetson-Nano-Kit-desarrollo/dp/B084DSDDLT

[39] Downey, A.B. (2012). *Think Python*. O'Reilly Media.

[40] Python. (25 de junio de 2023). *Tkinter-Interface de Python para Tcl/Tk*. https://docs.python.org/es/3/library/tkinter.html

[41] Chollet, F. (2017). *Deep Learning with Python.* Manning.

[42] Simonyan, K., & Zisserman, A. (10 de abril de 2015). *VGG16 and VGG19*. Keras Applications. https://keras.io/api/applications/vgg/

[43] Visual Studio Code. (2015). *Code editing. Redifined.* https://code.visualstudio.com/

[44] Heintz, B. (17 de diciembre de 2018). Training a neural Network to Detect Gestures with OpenCV in Python. *Towards Data Science*. https://towardsdatascience.com/training-a-neural-network-to-detect-gestures-with-opencv-in-python-e09b0a12bdf1

[45] Kazi, A. (26 de diciembre de 2022). Reconocimiento de lenguaje de señas usando CNN y OpenCV. *Viaje de la curiosidad*. https://arshad-kazi.com/sign-language-recognition-using-cnn-and-opencv/

[46] Daoust, M. (26 de enero de 2022). *Aumento de datos.* TensorFlow. https://www.tensorflow.org/tutorials/images/data_augmentation?hl=es-419

[47] Tan, M., & Le, Q.V. (2020). *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. https://arxiv.org/pdf/1905.11946.pdf

[48] Mathworks. (31 de agosto de 2022). *Transfer learning para entrenar modelos de Deep Learning*. https://la.mathworks.com/discovery/transfer-learning.html

[49] IBM. (16 de marzo de 2023). *¿Qué son las redes neuronales recurrentes?* https://www.ibm.com/es-es/topics/recurrent-neural-networks

[50] Rubén Tercero [rubentercero869] (16 de julio de 2023). *GrabacionError* [Vídeo]. Youtube. https://youtu.be/buWd_tMELhE

[51] Rubén Tercero [rubentercero869] (16 de julio de 2023). *Capture* [Vídeo]. Youtube. https://youtu.be/gVkhNG-LNVA

[52] Python. (25 de junio de 2023). *Tkinter-Interface de Python para Tcl/Tk*. https://docs.python.org/3/library/tkinter.html

[53] Scikit Learn. (2 de junio de 2023). *Metrics and scoring: quantifying the quality of predictions*. https://scikit-learn.org/stable/modules/model_evaluation.html

[54] Rubén Tercero [rubentercero869] (16 de julio de 2023). *GrabacionCuadro*. [Vídeo]. Youtube. https://youtu.be/YRyVl3Ctkac

[55] Rubén Tercero [rubentercero869] (27 de julio de 2023). *Pepe.* [Video]. Youtube. https://youtu.be/XY8dSijHme8

[56] Rubén Tercero [rubentercero869] (16 de julio de 2023). *GrabacionArantxa*. [Vídeo]. Youtube. https://youtu.be/wFaQdBQhzAk

[57] Python. (25 de junio de 2023). *Time Access and conversions*. https://docs.python.org/3/library/time.html

[58] Rodola, G. (24 de julio de 2022). *Psutil documentation*. Psutil. https://psutil.readthedocs.io/en/latest/

[59] Rubén Tercero. (26 de julio de 2023). *Lenguaje de Signos Español*. Kaggle. https://www.kaggle.com/datasets/rub3ntercero/lenguaje-de-signos-espaol